Thèse n° 10 221

Probabilistic methods for neural combinatorial optimization

Présentée le 14 juillet 2023

Faculté des sciences et techniques de l'ingénieur
Laboratoire de traitement des signaux 2
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Nikolaos KARALIAS

École
polytechnique
fédérale
de Lausanne

2023

“There is some fiction in your truth, and some truth in your fiction.”

Acknowledgements

First, I would like to express my gratitude to my supervisor Prof. Pierre Vandergheynst for his hospitality and for providing all the support and freedom that I needed to complete my PhD. I would like to thank the members of my thesis committee Prof. Pascal Frossard, Prof. Nicolas Flammarion, Dr. Francis Bach, and Dr. Petar Veličković. I am grateful for their time and for the thought-provoking discussions about my work and about related topics in machine learning. Most importantly, I am immensely grateful to Andreas Loukas for giving me the opportunity to pursue my PhD here at EPFL and for being the best supervisor and mentor I could have hoped for. I feel extremely lucky and honored to have had the chance to work with you. Your attitude towards research and your work ethic has had a profound impact on me, especially during the early formative years of my PhD. Outside of research, it was also great fun hanging out (thanks to Dora for the hospitality as well!), climbing, etc. Thank you.

Next, I sincerely thank Stefanie Jegelka and the members of her lab for the fruitful collaboration that we had throughout my PhD and for welcoming me to the lab and their weekly meetings as if I was a member of the group. I specifically want to say thank you to Derek, Hannah, Josh, and Sharut for being so welcoming and kind when we first met in person at NeurIPS2022. It was my first in-person conference and one of the most memorable experiences of my PhD because of you. Extra thanks to Josh for working with me and always being open-minded and willing to explore new ideas for our projects.

I want to extend my thanks to the current and former LTS2 members Ali, Anais, Anna, Benjamin, Daniel, Jakob, Jirka, Kostas, Louise, Mattia, Michael, Nicolas, Pengkang, Vamsi, Maria, Volodymyr, and Young Joo. In particular, I want to thank Adam, Daniele, Helena, and Giorgos for all the fun times we shared together as officemates. I am especially thankful for Giorgos' radio station choices that made the office experience so much better. Special thanks to Anne for helping me with practically every issue I had to deal with in Lausanne.

Apart from the members of my lab, there are a few more people that I want to mention. Whether it was for hanging out and BBQs, doing sports together, helping me with work, or just being a nice person to chat with in the corridor, I thank the following people: Abdellah, Alessandro, Antonia, Beril, Cedric, Cesare, Clement, Eda, Ersi, Eva, Flavio, Grigoris, Hermina, Irene, Isabel, Kostas, Marwa, Melivoia, Michela, Mireille, Nikos, Pinar, Roberto, Selina, Sina, Spiros, Stratis, Tolis, Vaggelis, Valentina.

I am particularly glad that I met Berta, Guillermo, and William. It has been great to spend time with you and to get to know you. You have also been the best climbing buddies I could ask for. On that note, I want to thank my parkour buddy John for all the great training sessions we had and for motivating me to keep pushing myself. I am also grateful for the support of my friends in Greece and other countries abroad: Antonis, Foteini, Vaggelis, and Vassilis as well as my old parkour buddies Giannis, Tzimis, Lefteris, and Nikos who have still stayed in touch.

As a last note to some friends, I want to thank my beloved ninjas Alex, Antreas, Giannis, Stratos,

Thanos, and Kostas. Their funny messages in our online group chat have consistently put a smile on my face and lifted my mood over the past 4 years.
Finally, I want to thank my family for their unwavering support and trust in me throughout the course of my PhD. All of this would not have been possible without you.

Lausanne, June 29, 2023 Nikolaos Karalias

Abstract

The monumental progress in the development of machine learning models has led to a plethora of applications with transformative effects in engineering and science. This has also turned the attention of the research community towards the pursuit of constructing artificial intelligence (AI) models with general reasoning capabilities. Yet, despite the staggering success of artificial neural networks in a variety of tasks that involve language and image generation or object detection and recognition, tasks that involve discrete and combinatorial problem-solving are still a fundamental blind spot of those models and present a longstanding obstacle in the road to general-purpose AI systems.

Combinatorial optimization problems are prominent representatives in that category as they present fundamental challenges that are hard to tackle within the standard machine learning paradigm. Two fundamental obstacles in this pursuit are i) the difficulty of navigating exponentially large discrete configuration spaces using continuous gradient-based optimization, ii) our inability to procure large amounts of labeled data due to the high computational budget that this requires. The subject of this thesis will be to develop a coherent approach to combinatorial optimization with neural networks that focuses on directly tackling those challenges.

In the first half of the thesis, we will present our proposal for neural combinatorial optimization without supervision. We demonstrate how it is possible to design continuous loss functions for constrained optimization problems in a way that enables training without access to labeled data. We leverage the celebrated probabilistic method from the field of combinatorics to argue about the existence of high-quality solutions within the learned representations of a neural network that has been trained with our approach. We also show how to deterministically recover those solutions using derandomization techniques from the literature.

In the second half, we expand the scope of our inquiry and design a general framework for continuous extensions of set functions. This approach enables training neural networks for discrete problems even when the objective and the constraints of the problem are given as a black box. We develop extensions for domains like the hypercube but also higher-dimensional domains like the cone of positive semi-definite matrices. This framework enables us to efficiently incorporate problem-specific priors in the pipeline which leads to improved empirical results. Finally, we show that the versatility of this approach extends beyond combinatorial optimization as it can be used to define a novel continuous surrogate of the discrete training error for classification problems. Overall, our proposed methods make progress in advancing the state of the art for neural combinatorial optimization through principled loss function design. Furthermore, by enabling the use of discrete functions in end-to-end differentiable models they pave the way for improved combinatorial and reasoning capabilities for machine learning algorithms.

Key Words: Combinatorial optimization, probabilistic method, unsupervised learning, set function extensions, learning in higher dimensions

Résumé

Les progrès monumentaux réalisés dans le développement de modèles d'apprentissage automatique ont conduit à une pléthore d'applications ayant des effets transformateurs dans les domaines de l'ingénierie et de la science. Cela a également attiré l'attention de la communauté des chercheurs sur la construction de modèles d'intelligence artificielle (IA) dotés de capacités de raisonnement générales. Cependant, malgré le succès retentissant des réseaux neuronaux dans une variété de tâches qui impliquent comme la génération de langage et d'images ou encore la détection et la reconnaissance d'objets, les tâches qui impliquent la résolution de problèmes discrets et combinatoires constituent toujours un angle mort fondamental de ces modèles. Ils représentent depuis longtemps un obstacle de longue date sur la voie des systèmes d'intelligence artificielle à usage général. Les problèmes d'optimisation combinatoire sont éminemment représentatifs de cette catégorie, car ils posent des défis fondamentaux difficiles à relever dans le cadre du paradigme standard de l'apprentissage automatique. Deux obstacles fondamentaux dans cette quête sont i) la difficulté de naviguer dans des espaces de configuration discrets exponentiellement grands en utilisant l'optimisation continue basée sur le gradient, ii) notre incapacité à obtenir de grandes quantités de données annotées en raison du coût de calcul élevé que cela nécessite. Le sujet de cette thèse sera de développer une approche cohérente de l'optimisation combinatoire avec des réseaux de neurones qui se concentre sur la résolution directe de ces défis. Dans la première moitié de la thèse, nous présenterons notre proposition pour l'optimisation combinatoire neuronale sans supervision. Nous démontrons comment il est possible de concevoir des fonctions de perte continues pour des problèmes d'optimisation contraints afin de permettre l'apprentissage sans accès à des données étiquetées. Nous nous appuyons sur la célèbre méthode probabiliste du domaine de la combinatoire pour argumenter l'existence de solutions de haute qualité dans les représentations apprises d'un réseau neuronal qui a été entraîné avec notre approche. Nous montrons également comment récupérer ces solutions de manière déterministe à l'aide de techniques de dérandomisation issues de la littérature. Dans la deuxième partie, nous élargissons le champ de notre enquête et concevons un cadre général pour les extensions continues des fonctions ensemblistes. Cette approche permet d'entraîner des réseaux neuronaux pour des problèmes discrets, même lorsque l'objectif et les contraintes du problème sont donnés sous forme de boîte noire. Nous développons des extensions pour des domaines tels que l'hypercube, mais aussi des domaines de plus haute dimension tels que le cône des matrices semi-définies positives. Nous montrons que ce cadre nous permet d'incorporer des éléments a priori spécifiques au problème dans le pipeline et améliore considérablement sa polyvalence puisqu'il peut être utilisé pour générer un nouveau substitut continu pour de l'erreur d'apprentissage dans la classification d'images. Dans l'ensemble, les méthodes que nous proposons font progresser l'état de l'art en matière d'optimisation combinatoire neuronale grâce à la conception de fonctions de perte fondées sur des principes. En outre, en

permettant l'utilisation de fonctions discrètes dans des modèles différentiables de bout en bout, elles ouvrent la voie à des capacités combinatoires et de raisonnement améliorées pour les algorithmes d'apprentissage automatique.
Mots clés : Optimisation combinatoire, Méthode probabiliste, Apprentissage non supervisé, Extension des fonctions de l'ensemble, apprentissage en haute dimension

CONTENTS

1 Introduction

1.1 Combinatorial Optimization and Machine Learning

Combinatorial optimization (CO) involves the study of problems where the goal is to find the best possible configuration out of a finite collection of items. This includes problems like boolean satisfiability (SAT), maximum independent set (MIS), and traveling salesperson (TSP). Combinatorial optimization has important real-world applications including circuit design and verification (Vizel et al., 2015), route scheduling (Toth and Vigo, 2002), communication networks (Cheng et al., 2006) and molecular chemistry (Ehrlich and Rarey, 2011). The study of combinatorial optimization problems has also had a tremendous impact in modern computer science and mathematics. It has elucidated deep connections between (convex) geometry, graph theory, optimization theory, and combinatorics (Lovász, 2019; Ardila, 2021). Combinatorial problems have also been instrumental in formalizing notions of efficient computation through the complexity class P (Edmonds, 1965), as well as developing central notions of computational complexity theory like NP-Completeness (Karp, 1972; Cook, 1971).

1.1.1 Problem setting

CO problems over a set that consists of n items typically involve finding a *feasible* subset $S \subseteq [n]$ that minimizes (or maximizes) an objective function $f : 2^{[n]} \to \mathbb{R}$. Here, $[n]$ denotes the set that contains all n items and $2^{[n]}$ denotes its power set. The collection of all feasible subsets of $[n]$ will be denoted by $\Omega \subseteq 2^{[n]}$. This type of problem is known as a *constrained optimization problem*. As an example, consider the maximum clique problem where we are given an undirected graph on n nodes $G = (V, E)$, with V denoting a finite set of nodes and $E \subseteq V \times V$ the set of edges where each edge is a tuple of nodes. Then, Ω is defined as the collection of all possible cliques in G, i.e., subsets $S \subseteq V$ such that, for every S, all pairs of nodes in S are connected by an edge in G. The goal is then to find the clique $S \in \Omega$ that has the largest number of nodes.

While several CO problems fit this description, it is also common to encounter CO problems without an objective function. In those cases, it is sufficient to find any feasible solution to the problem. The class of *constraint satisfaction problems (CSPs)* are precisely of this type. We are given a set of variables and a set of constraints on those variables. The goal is to find an assignment of values to the variables that satisfies all constraints. A classic example of a constraint satisfaction problem is SAT. In the SAT problem, we are given a logic formula that consists of Boolean variables (i.e., taking values “true” or “false”) and logical operations between them (AND, OR, NOT). The goal is then to find an assignment of values to the variables such that the entire formula evaluates to "true". If there exists such an assignment, the formula is said to be satisfiable. The logical operations present in the formula naturally constrain the feasible combinations of values that can be assigned to the variables. In this problem, the notion of feasibility is equivalent to the notion of satisfiability, i.e., Ω may be defined as the collection of variable assignments that evaluate to "true". Constraint satisfaction problems include games like Sudoku, graph coloring, and map coloring problems, as well as crossword and logic puzzles. It is worth mentioning that constraint satisfaction problems can be intimately connected to unconstrained optimization problems. Indeed, for any CSP instance, one may consider the optimization problem of maximizing the number of constraints that are being satisfied, or equivalently, minimizing the number of violated constraints. For a more detailed discussion on CSPs and their application see Brailsford et al. (1999).

1.1.2 Solving CO problems

While important combinatorial problems like the matching problem can be solved efficiently (i.e., are in P), many of the flagship combinatorial optimization problems are known to be NP-Hard. This means no known polynomial time algorithm may solve arbitrary instances of those problems. Therefore, a plethora of methods have been developed for different combinatorial problems. Depending on the type of constraints and the type of objective that a given problem has, different approaches may be adopted to solve it. For example, given a linear cost function and linear constraints on the variables of a constrained optimization program, one may employ a linear programming solver. More broadly, we may classify approaches to solving CO problems into the following three categories (Festa, 2014):

- Exact methods. These methods provide guarantees for the optimality of the solution. Branch and bound algorithms (Lodi, 2010) are a prominent example. They partition the problem into smaller relaxed subproblems and by iteratively solving each subproblem a better bound on the solution of the problem is obtained which progressively narrows down the solution space.
- Approximation algorithms (Williamson and Shmoys, 2011). These methods cannot guarantee optimality but they can guarantee that their solutions will be at most a given factor away from the optimal. Randomized algorithms and convex relaxations often fall in this category. A celebrated example is the (Goemans and Williamson, 1995) algorithm for the maximum cut problem which returns cuts with expected weight at least 0.87x the size of the optimal cut.
- (Meta)-Heuristics (Potvin and Gendreau, 2018). Those are algorithms that are efficient in practice but do not provide any solution quality guarantees. This category includes methods like tabu search (Glover, 1986) and genetic algorithms (Kramer, 2017).

Despite the large variety of methods that have been developed for solving those problems it is still relatively easy to generate hard instances that will be challenging for most state-of-the-art algorithms. Examples include random synthetic cryptographic instances for SAT (Ganesh and Vardi, 2020), and hard synthetic instances for other constraint satisfaction problems such as coloring and independent set (Prosser, 2012; Xu et al., 2007; Xu, 2007). However, as we will explain in the following paragraphs, those problems can still be carefully dissected and efficient solutions can be found for many problem instances that are encountered in the real world.

1.1.3 Motivation: "Nature is not an adversary"

While NP (or NP-Hard) problems are expected to be difficult, worst-case complexity can be misleading. A famous example is that of the simplex algorithm, for which worst-case examples were constructed and shown to take exponential time (Klee and Minty, 1972). Nevertheless, the performance of the algorithm is known to be fast in practice and this discrepancy between worst-case and average-case performance for the simplex algorithm has been extensively studied and explained (Spielman and Teng, 2004). Conversely, it is to be expected that for any hard problem, one can identify families of instances with special properties that make the problem tractable. In fact, in certain cases, it is possible to even prove that the problem is polynomial-time solvable for a given family of instances. Examples of such cases include the maximum independent set which can be computed exactly for perfect graphs in polynomial time (Grötschel et al., 1981) by calculating the theta number (Lovász, 1979), and SAT instances with dependency graphs of sufficiently bounded maximum degree which makes them solvable in polynomial time with a simple local resampling algorithm (Moser and Tardos, 2010). It is said that "nature is not an adversary" (Cappart et al., 2021a) because real-world instances often come with such additional structure that is imposed by the domain that generates them. Several such cases have been documented in the literature. For example, the SAT problem is NP-Complete and yet SAT Solvers perform "unreasonably well" in various sets of industrial instances (Ganesh and Vardi, 2020), often due to the high modularity that is present in their dependency structure (Ansótegui et al., 2012). Another interesting example is that of the maximum clique problem, which is often solved fast in practice for large real-world graphs due to their small clique-core gap (Walteros and Buchanan, 2020). These examples suggest that with sufficient expertise, one may identify the properties of the data distribution that make a hard problem tractable for a given set of instances and pick a suitable efficient algorithm for it. This serves as a central motivation for *neural combinatorial optimization*, i.e., leveraging the abilities of machine learning models to identify patterns and relevant features in data in order to solve combinatorial optimization problems.

1.1.4 Complementary perspectives on ML and CO

Machine learning (ML) aims to solve problems through a *data-driven* approach: the parameters of a model are updated to capture the properties of available observations with the purpose of predicting properties of unseen data or even generating new data. In the past decade, this approach has had a striking impact in several fields of science (Jumper et al., 2020;

Larkoski et al., 2020) and engineering (Zhang and Yu, 2020; Fawzi et al., 2022) as well as in industrial applications (Koumchatzky and Andryeyev, 2017; Ying et al., 2018). The most recent example of the success of learning architectures is found in large language models (LLMs) (Bommasani et al., 2021) like ChatGPT, which have demonstrated impressive capabilities in numerous tasks including coding problems, interaction with humans, and complex games (Karpas et al., 2022; Schick et al., 2023; Bubeck et al., 2023). In this thesis, our goal will be to use machine learning models to solve combinatorial optimization problems. To better understand the motivation behind this, we discuss two complementary perspectives on the topic. They both highlight the potential benefits of the fusion between combinatorial optimization and machine learning.

The ability of machine learning models to capture relevant patterns in data and exploit them for downstream tasks motivates their use in combinatorial optimization. The powerful feature extraction capabilities of neural networks may circumvent the computational hardness barrier of combinatorial problems by identifying and exploiting the structure that is present in real-world instances. In that sense, they may prove helpful in the improvement of existing algorithms and the discovery of new ones, therefore advancing the state of the art in the field. Examples include the breakthroughs in strategy games like chess and GO (Schrittwieser et al., 2020; Zhang and Yu, 2020) which combine trained models with Monte Carlo Tree Search (Coulom, 2007).

The converse is also true: the study and use of combinatorial optimization problems and algorithms in the context of machine learning can lead to improvements in the understanding of neural network architectures and the construction of AI models with reasoning and algorithmic capabilities. A famous example is the graph isomorphism problem which has been central in characterizing the expressive power of graph neural networks (GNNs) (Morris et al., 2019; Loukas, 2020a). Other related results include bounds on the approximation ratios of GNNs for several combinatorial problems on bounded degree graphs (Sato et al., 2020), impossibility results for computing structural properties and solving decision problems (Loukas, 2019), and the inability of graph neural networks to count substructures (Chen and Tian, 2019). These all highlight certain limitations of modern learning architectures when it comes to performing combinatorial tasks.

A crucial point regarding those two complementary perspectives is that progress in one direction does not directly entail progress in the other. For example, consider a model that learns to identify which heuristic to pick among a pool of heuristics for a given problem instance (Nudelman et al., 2004). That model may learn to function as a metaheuristic and improve the state-of-the-art results on a series of benchmarks. However, in that scenario, this pipeline may not generalize to problems or instances for which an efficient heuristic is not known. Therefore, the problem-solving abilities of such a pipeline are predicated on the existence of classical heuristics but not on an improved ability of the actual machine learning model to perform reasoning or combinatorial tasks. In the other direction, progress in the development of combinatorial and reasoning capabilities of neural networks may not directly translate to state-of-the-art results for CO. The top-performing algorithms for combinatorial problems are a product of painstaking engineering and clever combination of heuristics. It is likely that improvement in combinatorial capabilities alone will not be sufficient and any ML-driven algorithm for CO problems will have to be specifically engineered for the given task in order to be competitive. Although both directions are worth pursuing independently, meaningful progress will be certainly achieved in the “Goldilocks” zone between those two, where the study of neural architectures from a combinatorial lens and the pursuit of state-of-the-art CO results with the help of ML models will coalesce into architectures with improved reasoning and combinatorial capabilities that can advance the state of the art.

1.2 Central challenges of ML for CO

So far we have seen how work at the intersection of CO and ML can be a fruitful endeavor for both of those fields. This leads us to discuss the central challenges and developments in the literature that have motivated the contributions of this thesis.

1.2.1 Fusing discrete and continuous computation

Despite the impressive progress that has been made in flagship machine learning pipelines like LLMs, it is known that they can perform poorly when it comes to reasoning and planning (Valmeekam et al., 2022). Even for well-known computationally tractable problems like sorting small lists of numbers, ChatGPT will not consistently produce correct solutions [1]. Those failures are not the byproduct of computational limitations, as LLMs are trained on vast datasets and have access to powerful computational resources. More broadly, combinatorial and algorithmic problem-solving has been one of the most challenging obstacles to overcome, as ML models are still often outperformed by simple heuristics or exact algorithms (Boettcher, 2023a,b; Ciarella et al., 2023; Liu et al., 2022; Böther et al., 2022). This points to the significant limitations of prominent neural network pipelines in performing reasoning and combinatorial tasks. A fundamental challenge that underlies the lack of success in achieving those capabilities for neural networks is *the difficulty of fusing discrete computation and neural network architectures trained with gradient descent*. Neural networks generally operate on continuous and differentiable representations while discrete computations typically require discrete data and introduce non-differentiable computational nodes in the pipeline. Various techniques have been proposed that smoothly incorporate discrete operations like sampling (Huijben et al., 2022), sorting (Mena et al., 2018; Grover et al., 2018), convex optimization layers (Agrawal et al., 2019), and logic gates (Petersen et al., 2022) in neural network architectures. These often involve the use of stochastic gradient estimation (Grathwohl et al., 2018), reparametrization techniques, and continuous relaxations (Niculae et al., 2023). Alas, these techniques tend to be highly specialized. General-purpose reliable approaches for solving a wide range of existing

[1]As of April 2023, many failure modes for sorting can be found. A common example is to ask the model to sort several numbers, each represented as a fraction.

combinatorial and algorithmic problems are still not available. The focus of this thesis will be to make progress on this challenge by designing a framework that enables the use of discrete functions in differentiable machine learning pipelines.

1.2.2 Computational efficiency, generalization, and neural algorithmic reasoning

A critical consideration for machine learning models is their computational efficiency. Through the use of continuous proxies for discrete combinatorial objectives, we will show how we can train neural networks without any supervision. This eschews the need for labeled solutions which can be time-consuming for hard combinatorial problems, thereby increasing computational efficiency. The absence of labels can also naturally help to avoid overfitting, leading to better generalization. These are some of the key motivations for the work that we will present in Chapter 2. Since the publication of the work in Chapter 2, several other methods have been proposed in the literature that adopt a similar approach to solving combinatorial problems (Schuetz et al., 2022; Dai et al., 2020; Sun et al., 2022; Xu et al., 2020a; Min et al., 2022), which speaks to its viability.

Generalization has also motivated the study of inductive biases in machine learning models (Goyal and Bengio, 2022). It is commonly argued that due to the no free lunch theorem, it is impossible to have a general-purpose model that excels at all tasks; some degree of specialization is required. That is precisely the role of inductive biases, which are the innate assumptions that are built into the pipeline through various architectural and algorithmic choices in order to effectively restrict the hypothesis space (i.e., the space of possible functions the model can learn) (Baxter, 2000). A good example of this are graph neural networks, which are often successful in tasks that involve relational reasoning, precisely because they have a relational inductive bias (Battaglia et al., 2018). Therefore, it seems clear that providing some innate structure to models depending on the task will be conducive to better generalization. A formalization of this notion of compatibility between model biases and task structure comes from the work on algorithmic alignment (Xu et al., 2020b). A model is said to align with an algorithm if the model can efficiently learn to simulate that algorithm. Models which align well with known algorithms are expected to enjoy better sample complexity and generalization properties. Returning back to graph neural networks as an example, they are known to align with the Bellman-Ford algorithm for dynamic programming (Xu et al., 2020b; Dudzik and Veličković, 2022). This is backed up empirically as graph neural networks perform well on dynamic-programming problems like shortest paths on graphs. Another benefit of such alignment is that the internal operations of the model become more interpretable which may lead to performance guarantees.

Those considerations have motivated the research program on neural algorithmic reasoning (Veličković et al., 2020; Veličković et al., 2022) and related works that focus on the execution of algorithms with neural networks (Yan et al., 2020; Li et al., 2020). They also form the primary motivation behind the contribution of Chapter 3. Discrete operations and discrete functions form the building blocks of algorithms. The goal of our contribution in that chapter is to provide a way to incorporate such building blocks in neural network pipelines by presenting a general methodology for using discrete functions with the continuous high-dimensional features of neural networks.

1.2.3 Solving discrete problems in high dimensional spaces

The neural algorithmic reasoning blueprint places additional emphasis on the neural execution of algorithms natively on the learned high-dimensional representations of the neural network. This aims to avoid information bottlenecks in neural architectures and to take full advantage of their feature-extracting capabilities. The use of algorithms in that context will provide performance and reliability guarantees while maintaining the benefits of neural representations. Conceptually, this is related to fundamental ideas in approximation algorithms and graph theory which revolve around solving combinatorial problems on discrete structures by embedding them in suitable high-dimensional spaces. The geometric properties of the embeddings in that space are then used to determine the combinatorial properties of the discrete structure. For example, it has been shown that certain connectivity properties of undirected graphs can be determined by showing that their embeddings in a suitable high-dimensional space satisfy specific orthogonality and linear independence conditions (Lovász et al., 1989). Another major achievement is the use of semidefinite programming for approximation algorithms and in particular the celebrated Goemans-Williamson maximum cut algorithm (Goemans and Williamson, 1995), which relies on embedding a graph in a higher dimensional space and using a probabilistic argument to obtain a certificate for the size of the maximum cut that can be recovered. In the same spirit, Chapter 3 will show how we can use a discrete function directly on a continuous high-dimensional space by defining a suitable continuous counterpart of the discrete function in that space.

1.3 A general approach to neural combinatorial optimization

The goal of this thesis will be to provide tools that enable machine learning pipelines to successfully solve diverse combinatorial problems. Our methodology is directly motivated by the considerations around generalization and efficiency that we described in our discussion. To that end, we will propose a general framework for unsupervised neural combinatorial optimization. We will describe an approach to creating differentiable loss functions that can be used to train neural networks to solve combinatorial optimization problems on graphs without access to labels. This is primarily achieved through a loss function derived using the probabilistic method which represents the objective of the constrained optimization problem. The neural network learns a distribution over solutions from which we are able to deterministically recover a high-quality solution in a principled fashion. We show that this leads to strong experimental results against neural baselines, heuristics, and even general-purpose solvers. To demonstrate the generality of our approach we derive loss functions for numerous important combinatorial problems. Furthermore, we provide a detailed discussion of the main limitations of the method and explain how more sophisticated probabilistic techniques can be utilized to overcome them for the class of constraint satisfaction problems.

To tackle the more general problem of fusing discrete computation with neural networks, we propose a framework for constructing continuous extensions of discrete functions that are defined on sets. Extensions are continuous versions of the discrete functions that agree with them at discrete points and can be used as drop-in replacements in differentiable end-to-end pipelines. Our extensions can be viewed from a probabilistic perspective as efficiently computable expectations of distributions over discrete function evaluations. We describe several scalar extensions and provide guidelines on how to derive new ones. We also establish important properties of extensions that allow them to be used in differentiable end-to-end models. Motivated by ideas from approximation algorithms, semidefinite programming, and neural algorithmic reasoning, we show how to define extensions on higher dimensional spaces. This allows us to evaluate set functions on the high-dimensional representations learned by neural networks. We use extensions to define loss functions which leads to improved performance in combinatorial optimization. Additionally, we show how to build simple constraints into our extensions. Going beyond combinatorial optimization, we build an extension of the training error of a classifier for the purpose of image classification. In order to further expand the scope of extensions, we show how they can be defined on different geometries which further enhances their applicability. Finally, we will describe an extension-based probabilistic penalty loss that overcomes some of the major limitations of the losses described in Chapter 2.

1.4 Thesis outline

The rest of the thesis is organized as follows: In the second chapter, we describe our approach to unsupervised neural combinatorial optimization which uses the Erdős probabilistic method to construct differentiable loss functions for CO problems. We show how to obtain certificates of solution quality from the network and how to obtain an efficient deterministic algorithm that decodes discrete solutions from the learned distributions of the network. In the second half of the chapter, we provide an extended discussion of the applicability and the limitations of the method and discuss potential improvements.

In the third chapter, we build a general framework for learning with discrete functions in end-to-end differentiable pipelines. We do this by defining continuous extensions, i.e., continuous counterparts of discrete functions that agree with the original function on discrete points. We then proceed to define extensions on higher dimensional domains and test them experimentally on combinatorial optimization problems and classification problems. Furthermore, we provide an extended discussion on the properties of extensions, how to derive new ones, and how to generalize them to different geometries and different problems.

In the final chapter, we provide an overview of the contributions of the thesis and discuss some potential future directions.

List of contributions. This thesis is composed of the following published papers and articles:

- **Chapter 2:** Karalias, N. and Loukas, A. (2020). Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33:6659–6672.
- **Chapter 3:** Karalias, N., Robinson, J. D., Loukas, A., and Jegelka, S. (2022). Neural set function extensions: Learning with discrete functions in high dimensions. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

During my doctoral studies, I also co-authored the following paper, which I do not present in this document:

- Bouritsas, G., Loukas, A., Karalias, N., and Bronstein, M. (2021). Partition and code: learning how to compress graphs. *Advances in Neural Information Processing Systems*, 34:18603–18619.

2 Erdős Goes Neural: an unsupervised learning for combinatorial optimization on graphs

2.1 Introduction

Combinatorial optimization (CO) includes a wide range of computationally hard problems that are omnipresent in scientific and engineering fields. Among the viable strategies to solve such problems are neural networks, which were proposed as a potential solution by Hopfield and Tank (1985). Neural approaches aspire to circumvent the worst-case complexity of NP-hard problems by only focusing on instances that appear in the data distribution.

Since Hopfield and Tank, the advent of deep learning has brought new powerful learning models, reviving interest in neural approaches for combinatorial optimization. A prominent example is that of graph neural networks (GNNs) (Gori et al., 2005; Scarselli et al., 2008), whose success has motivated researchers to work on CO problems that involve graphs (Joshi et al., 2019a; Yolcu and Poczos, 2019; Khalil et al., 2016; Gasse et al., 2019; Lemos et al., 2019; Nowak et al., 2017; Bai et al., 2020; Prates et al., 2019) or that can otherwise benefit from utilizing a graph structure in the problem formulation (Toenshoff et al., 2019) or the solution strategy (Gasse et al., 2019). The expressive power of graph neural networks has been the subject of extensive research (Xu et al., 2019; Loukas, 2020a; Chen et al., 2020b; Sato et al., 2019; Sato, 2020; Barceló et al., 2019; Garg et al., 2020). Encouragingly, GNNs can be Turing universal in the limit (Loukas, 2020b), which motivates their use as general-purpose solvers.

Yet, despite recent progress, CO problems still pose a significant challenge to neural networks. Successful models often rely on supervision, either in the form of labeled instances (Li et al., 2018; Selsam et al., 2018; Joshi et al., 2019a) or of expert demonstrations Gasse et al. (2019). This success comes with drawbacks: obtaining labels for hard problem instances can be computationally infeasible (Yehuda et al., 2020), and direct supervision can lead to poor generalization (Joshi et al., 2019b). Reinforcement learning (RL) approaches have also been used for both classical CO problems (Chen and Tian, 2019; Yolcu and Poczos, 2019; Yao et al., 2019; Kool et al., 2018; Deudon et al., 2018; Khalil et al., 2017; Bai et al., 2020) as well as for games with large discrete action spaces, like Starcraft (Vinyals et al., 2019) and Go (Silver et al., 2017). However, not being fully-differentiable, they tend to be harder and more time consuming to train.

An alternative to these strategies is unsupervised learning, where the goal is to model the problem with a differentiable loss function whose minima represent the discrete solution to the combinatorial problem (Smith, 1999; Bianchi et al., 2019; Amizadeh et al., 2018, 2019; Toenshoff et al., 2019; Yao et al., 2019). Unsupervised learning is expected to aid in generalization, as it allows the use of large unlabeled datasets, and it is often envisioned to be the long term goal of artificial intelligence. However, in the absence of labels, deep learning faces practical and conceptual obstacles. Continuous relaxations of objective functions from discrete problems are often faced with degenerate solutions or may simply be harder to optimize. Thus, successful training hinges on empirically-identified correction terms and auxiliary losses (Bianchi et al., 2019; Amizadeh et al., 2019; Van den Bout and Miller, 1989). Furthermore, it is especially challenging to decode valid (with respect to constraints) discrete solutions from the soft assignments of a neural network (Li et al., 2018; Toenshoff et al., 2019), especially in the absence of complete labeled solutions (Selsam et al., 2018).

2.1.1 Contributions

Our framework aims to overcome some of the aforementioned obstacles of unsupervised learning: *it provides a principled way to construct a differentiable loss function whose minima are guaranteed to be low-cost valid solutions of the problem.* Our approach is inspired by Erdős' probabilistic method and entails two steps: First, we train a GNN to produce a distribution over subsets of nodes of an input graph by minimizing a probabilistic penalty loss function. Successfully optimizing our loss is guaranteed to yield good integral solutions that obey the problem constraints. After the network has been trained, we employ a well-known technique from randomized algorithms to sequentially and deterministically decode a valid solution from the learned distribution. The procedure is schematically illustrated in Figure 2.1.

We demonstrate the utility of our method in two NP-hard graph-theoretic problems: the *maximum clique* problem Bomze et al. (1999) and a *constrained min-cut* problem Bruglieri et al. (2004); Svitkina and Fleischer (2011) that can perform local graph clustering (Andersen et al., 2006; Wang et al., 2017). In both cases, our method achieves competitive results against neural baselines, discrete algorithms, and mathematical programming solvers. Our method outperforms the CBC solver (provided with Google's OR-Tools), while also remaining competitive with the SotA commercial solver Gurobi 9.0 (Gurobi Optimization, 2020) on larger instances. Finally, our method outperforms both neural baselines and well-known local graph clustering algorithms in its ability to find sets of good conductance, while maintaining computational efficiency. [1]

2.1.2 Related work

Most neural approaches to CO are supervised. One of the first modern neural networks were the Pointer Networks (Vinyals et al., 2015), which utilized a sequence-to-sequence model for the travelling salesman problem (TSP). Since then, numerous works have combined GNNs

[1]Code available at: https://github.com/Stalence/erdos_neu

with various heuristics and search procedures to solve classical CO problems, such as quadratic assignment Nowak et al. (2017), graph matching (Bai et al., 2018), graph coloring (Lemos et al., 2019), TSP (Li et al., 2018; Joshi et al., 2019a), and even sudoku puzzles (Palm et al., 2018). Another fruitful direction has been the fusion with solvers. For example, Neurocore (Selsam and Bjørner, 2019) incorporates an MLP to a SAT solver to enhance variable branching decisions, whereas Gasse et al. (2019) learn branching approximations by a GNN and imitation learning. Further, Wang et al. (2019) include an approximate SDP satisfiability solver as a neural network layer and Vlastelica et al. (2019) incorporate exact solvers within a differentiable architecture by smoothly interpolating the solver's piece-wise constant output. Unfortunately, the success of supervised approaches hinges on building large training sets with already solved hard instances, resulting in a chicken and egg situation. Moreover, since it is hard to efficiently sample unbiased and representative labeled instances of an NP-hard problem (Yehuda et al., 2020), labeled instance generation is likely not a viable long-term strategy either.

Training neural networks without labels is generally considered to be more challenging. One possibility is to use RL: Khalil et al. (2017) combine Q-Learning with a greedy algorithm and structure2vec embeddings to solve max-cut, minimum vertex cover, and TSP. Q-Learning is also used in Bai et al. (2020) for the maximum common subgraph problem. On the subject of TSP, the problem was also solved with policy gradient learning combined with attention Kool et al. (2018); Deudon et al. (2018); Bello et al. (2016). Attention is ubiquitous in problems that deal with sequential data, which is why it has been widely used with RL for the problem of vehicle routing Gao et al. (2020); Nazari et al. (2018); Peng et al. (2019); James et al. (2019). Another interesting application of RL is the work of Yolcu and Poczos (2019), where the REINFORCE algorithm is employed in order to learn local search heuristics for the SAT problem. This is combined with curriculum learning to improve stability during training. Finally, Chen and Tian (2019) use actor-critic learning to iteratively improve complete solutions to combinatorial problems. Though a promising research direction, deep RL methods are far from ideal, as they can be sample inefficient and notoriously unstable to train—possibly due to poor gradient estimates, dependence on initial conditions, correlations present in the sequence of observations, bad rewards, sub-optimal hyperparameters, or poor exploration (Thrun and Schwartz, 1993; Nikishin et al., 2018; Irpan, 2018; Mnih et al., 2015).

The works that are more similar to ours are those that aim to train neural networks in a differentiable and end-to-end manner: Toenshoff et al. (2019) model CO problems in terms of a constraint language and utilize a recurrent GNN, where all variables that coexist in a constraint can exchange messages. Their model is completely unsupervised and is suitable for problems that can be modeled as maximum constraint satisfaction problems. For other types of problems, like independent set, the model relies on empirically selected loss functions to solve the task. Amizadeh et al. (2018, 2019) train a GNN in an unsupervised manner to solve the circuit-SAT and SAT problems by minimizing an appropriate energy function. Finally, Yao et al. (2019) train a GNN for the max-cut problem on regular graphs without supervision by optimizing a smooth relaxation of the cut objective and policy gradient.

Our approach innovates from previous works in the following ways: it enables training a neural network in an unsupervised, differentiable, and end-to-end manner, while also ensuring that identified solutions will be integral and will satisfy problem constraints. Crucially, this is achieved in a simple and mathematically-principled way, without resorting to continuous relaxations, regularization, or heuristic corrections of improper solutions. In addition, our approach does not necessitate polynomial-time reductions, but solves each problem directly.

2.1.3 Background: the probabilistic method

The probabilistic method is a nonconstructive proof method pioneered by Paul Erdős. It is used to demonstrate the existence of objects with desired combinatorial properties (Alon and Spencer, 2004; Erdös, 1959; Szegedy, 2013) but has also served as the foundation for important algorithms in the fields of computer science and combinatorial optimization (Moser and Tardos, 2010; Raghavan, 1988).

Let us consider the common didactic example of the maximum cut problem on a simple undirected graph (Mitzenmacher and Upfal, 2017). The goal is to bipartition the nodes of the graph in such a way that the number of edges with endpoints in both partitions (i.e., the cardinality of the cut-set) is maximized. For simplicity we will refer to the cardinality of the cut-set as the cut. Suppose we decide the bipartition based on a fair coin flip, i.e., we split the nodes of the graph by assigning them to a heads or a tails set. An edge belongs to the cut-set when its endpoints belong to different sets. This happens with probability $1/2$, which implies that *the expected cut* will be equal to half of the edges of the graph. Thus, by Markov's inequality and given that the cut is non-negative, it follows that there exists a bipartitioning that contains *at least half* of the edges of the graph.

To obtain such a solution deterministically, we will utilize the method of conditional expectation (Raghavan, 1988): we sequentially visit every node v_i in the graph and we compute the expected cut conditioned on v_i belonging to the heads or tails set (together with all the decisions made until the i-th step) and add v_i to the set (heads or tails) that yields smaller conditional expected cut. Since the (conditional) expectation can only improve at every step, the sets recovered are guaranteed to cut at least half the edges of the graph, as proved earlier.

Our goal is to re-purpose this classic approach to tackle combinatorial optimization problems with deep learning. Instead of using a naive probability assignment like in the maxcut example, the probability distribution is learned by a GNN which allows us to obtain higher quality solutions. Additionally, we show how this argument may be extended to incorporate constraints within the learning paradigm.

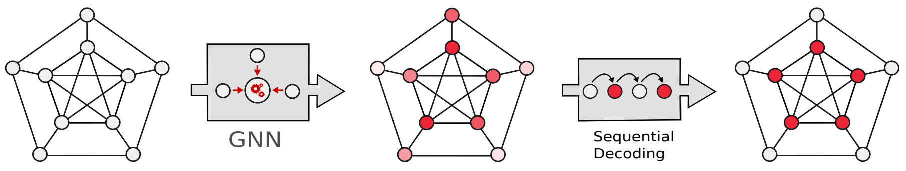


Figure 2.1: Illustration of the "Erdős goes neural" pipeline. First, a differentiable loss is derived for a given problem using the probabilistic method. Next, a GNN is trained in an unsupervised way using the derived loss to output a probability distribution over the nodes, essentially providing a probabilistic certificate for the existence of a low cost feasible solution. At inference time, a discrete solution satisfying the certificate is obtained in a sequential and deterministic manner by the method of conditional expectation.

2.2 The Erdős probabilistic method for deep learning

We focus on combinatorial problems on weighted graphs $G = (V, E, w)$ that are modelled as constrained optimization problems admitting solutions that are node sets:

$$\min_{S \subseteq V} f(S; G) \quad \text{subject to} \quad S \in \Omega. \tag{2.1}$$

Above, Ω is a family of sets having a desired property, such as forming a clique or covering all nodes. This yields a quite general formulation that can encompass numerous classical graph-theoretic problems, such as the maximum clique and minimum vertex cover problems.

2.2.1 The "Erdős Goes Neural" pipeline

Rather than attempting to optimize the non-differentiable problem equation 2.1 directly, we propose to train a GNN to identify distributions of solutions with provably advantageous properties. Our approach is inspired by Erdős' probabilistic method, a well known technique in the field of combinatorics that is used to prove the existence of an object with a desired combinatorial property.

As visualized in Figure 2.1, our method consists of three steps:

1. Construct a GNN g_θ that outputs a distribution $\mathcal{D} = g_\theta(G)$ over sets.
2. Train g_θ to optimize the probability that there exists a valid $S^* \sim \mathcal{D}$ of small cost $f(S^*; G)$.
3. Deterministically recover S^* from $\mathcal{D}$ by the method of conditional expectation.

There are several possibilities in instantiating $\mathcal{D}$. We opt for the simplest and suppose that the decision of whether $v_i \in S$ is determined by a Bernoulli random variable x_i of probability p_i. The network can trivially parametrize $\mathcal{D}$ by computing p_i for every node v_i. Keeping the distribution simple will aid us later on to tractably control relevant probability estimates.

Next, we discuss how g_θ can be trained (Section 2.2.2) and how to recover S^* from $\mathcal{D}$ (Section 2.2.3).

2.2.2 Deriving a probabilistic loss function

The main challenge of our method lies in determining how to tractably and differentiably train g_θ. Recall that our goal is to identify a distribution that contains low-cost and valid solutions.

The probabilistic loss

Aiming to build intuition, let us first consider the unconstrained case. To train the network, we construct a loss function $\ell(\mathcal{D};G)$ that abides to the following property:

$$P(f(S;G) < \ell(\mathcal{D};G)) > t \quad \text{with} \quad \mathcal{D} = g_\theta(G). \tag{2.2}$$

Any number of tail inequalities can be used to instantiate such a loss, depending on the structure of f. If we only assume that f is non-negative, Markov's inequality yields

$$\ell(\mathcal{D};G) \triangleq \frac{\mathbb{E}\left[f(S;G)\right]}{1-t} \quad \text{for any} \quad t \in [0,1). \tag{2.3}$$

If the expectation cannot be computed in closed-form, then any upper bound also suffices.

The main benefit of approaching the problem in this manner is that the surrogate (and possibly differentiable) loss function $\ell(\mathcal{D};G)$ can act as a certificate for the existence of a good set in the support of $\mathcal{D}$. To illustrate this, suppose that one has trained g_θ until the loss is sufficiently small, say $\ell(\mathcal{D};G) = \epsilon$. Then, by the probabilistic method, there exists with strictly positive probability a set S^* in the support of $\mathcal{D}$ whose cost $f(S^*;G)$ is at most ϵ.

The probabilistic penalty loss

To incorporate constraints, we take inspiration from penalty methods in constrained optimization and add a term to the loss function that penalizes deviations from the constraint.

Specifically, we define the probabilistic penalty function $f_p(S;G) \triangleq f(S;G) + \mathbf{1}_{S\notin\Omega}\,\beta$, where β is a scalar. The expectation of f_p yields the probabilistic penalty loss:

$$\ell(\mathcal{D};G) \triangleq \mathbb{E}\left[f(S;G)\right] + P(S \notin \Omega)\,\beta. \tag{2.4}$$

We prove the following:

Theorem 1. Fix any $\beta > \max_S f(S;G)$ and let $\ell(\mathcal{D};G) < (1-t)\beta$. With probability at least t, set $S^* \sim \mathcal{D}$ satisfies

$$f(S^*;G) < \ell(\mathcal{D};G)/(1-t) \;\text{ and }\; S^* \in \Omega,$$

under the condition that f is non-negative.

Hence, similar to the unconstrained case, the penalized loss acts as a certificate for the existence of a low-cost set, but now the set is also guaranteed to abide to the constraints Ω. The main requirement for incorporating constraints is to be able to differentiably compute an upper estimate of the probability $P(S \notin \Omega)$. A worked out example of how $P(S \notin \Omega)$ can be controlled is provided in Section 2.3.1.

The special case of linear box constraints

An alternative construction can be utilized when problem equation 2.1 takes the following form:

$$\min_{S \subseteq V} f(S;G) \quad \text{subject to} \quad \sum_{v_i \in S} a_i \in [b_l, b_h], \tag{2.5}$$

with a_i, b_l, and b_h being non-negative scalars.

We tackle such instances with a two-step approach. Denote by $\mathcal{D}^0$ the distribution of sets predicted by the neural network and let $p_1^0, \ldots, p_n^0$ be the probabilities that parametrize it. We rescale these probabilities such that the constraint is satisfied in expectation:

$$\sum_{v_i \in V} a_i p_i = \frac{b_l + b_h}{2}, \quad \text{where} \quad p_i = \text{clamp}\big(c\, p_i^0, 0, 1\big) \quad \text{and} \quad c \in \mathbb{R}.$$

Though non-linear, the aforementioned feasible re-scaling can be carried out by a simple iterative scheme (detailed in Section A.3). If we then proceed as in Section 2.2.2 by utilizing a probabilistic loss function that guarantees the existence of a good unconstrained solution, we have:

Theorem 2. Let $\mathcal{D}$ be the distribution obtained after successful re-scaling of the probabilities. For any (unconstrained) probabilistic loss function that abides to $P(f(S;G) < \ell(\mathcal{D};G)) > t$, set $S^* \sim \mathcal{D}$ satisfies $f(S^*;G) < \ell(\mathcal{D};G)$ and $\sum_{v_i \in S^*} a_i \in [b_l, b_h]$, with probability at least $t - 2\exp\big(-(b_h - b_l)^2 / \sum_i 2a_i^2\big)$.

Section 2.3.2 presents a worked-out example of how Theorem 2 can be applied.

2.2.3 Retrieving integral solutions

A simple way to retrieve a low cost integral solution S^* from the learned distribution $\mathcal{D}$ is by monte-carlo sampling. Then, if $S^* \sim \mathcal{D}$ with probability t, the set can be found within the first k samples with probability at least $1 - (1-t)^k$. However, our goal is to deterministically obtain S^* so we will utilize the method of conditional expectation that was introduced in Section 2.1.3.

Let us first consider the unconstrained case. Given $\mathcal{D}$, the goal is to identify a set S^* that satisfies $f(S^*;G) \le \mathbb{E}\left[f(S;G)\right]$. To achieve this, one starts by sorting $v_1, \ldots, v_n$ in order of decreasing probabilities p_i. Let $S_{\text{reject}} = \varnothing$ be the set of nodes not accepted in the solution. Set $S^* = \varnothing$ is then iteratively updated one node at a time, with v_i being included to S^* in the i-th step if $\mathbb{E}\left[f(S;G) \mid S^* \subset S,\ S \cap S_{\text{reject}} = \varnothing,\ \text{and } v_i \in S\right] < \mathbb{E}\left[f(S;G) \mid S^* \subset S,\ S \cap S_{\text{reject}} = \varnothing,\ \text{and } v_i \notin S\right]$. This sequential decoding works because the conditional expectation never increases.

In the case of the probabilistic penalty loss, the same procedure is applied w.r.t. the expectation of $f_p(S;G)$. The latter ensures that the decoded set will match the claims of Theorem 1. For the method of Section 2.2.2, a sequential decoding can guarantee either that the cost of $f(S^*;G)$ is small or that the constraint is satisfied.

2.3 Case studies

This section demonstrates how our method can be applied to two well known NP-hard problems: the *maximum clique* (Bomze et al., 1999) and the *constrained minimum cut* (Bruglieri et al., 2004) problems.

2.3.1 The maximum clique problem

A clique is a set of nodes such that every two distinct nodes are adjacent. The maximum clique problem entails identifying the clique of a given graph with the largest possible number of nodes:

$$\min_{S \subseteq V} -w(S) \quad \text{subject to} \quad S \in \Omega_{\text{clique}}, \tag{2.6}$$

with Ω_{clique} being the family of cliques of graph G and $w(S) = \sum_{v_i, v_j \in S} w_{ij}$ being the weight of S. Optimizing $w(S)$ is a generalization of the standard cardinality formulation to weighted graphs. For simple graphs, both weight and cardinality formulations yield the same minimum.

We can directly apply the ideas of Section 2.2.2 to derive a probabilistic penalty loss:

Corollary 1. Fix positive constants γ and β satisfying $\max_S w(S) \le \gamma \le \beta$ and let $w_{ij} \le 1$. If

$$\ell_{\text{clique}}(\mathcal{D};G) \triangleq \gamma - (\beta+1) \sum_{(v_i,v_j)\in E} w_{ij} p_i p_j + \frac{\beta}{2} \sum_{v_i \ne v_j} p_i p_j < (1-t)\beta,$$

then, with probability at least t, set $S^* \sim \mathcal{D}$ is a clique of weight $w(S^*) > \gamma - \ell_{\text{clique}}(\mathcal{D};G)/(1-t)$.

The loss function ℓ_{clique} can be evaluated in linear time w.r.t. the number of edges of G by rewriting the rightmost term as $\sum_{v_i \ne v_j} p_i p_j = (\sum_{v_i \in V} p_i)^2 - \sum_{(v_i,v_j)\in E} 2 p_i p_j$.

A remark. One may be tempted to fix $\beta \to \infty$, such that the loss does not feature any hyperparameters. However, with mini-batch gradient descent it can be beneficial to tune the contribution of the two terms in the loss to improve the optimization. This was also confirmed in our experiments, where we selected the relative weighting according to a validation set.

Decoding cliques. After the network is trained, valid solutions can be decoded sequentially based on the procedure of Section 2.2.3. The computation can also be sped up by replacing conditional expectation evaluations (one for each node) by a suitable upper bound. Since the clique property is maintained at every point, we can also efficiently decode cliques by sweeping nodes (in the order of larger to smaller probability) and only adding them to the set when the clique constraint is satisfied.

2.3.2 Graph partitioning

The simplest partitioning problem is the minimum cut: find set $S \subset V$ such that $\text{cut}(S) = \sum_{v_i \in S,\ v_j \notin S} w_{ij}$ is minimized. Harder variants of partitioning aim to provide control on partition balance, as well as cut weight. We consider the following constrained min-cut problem:

$$\min_S \text{cut}(S) \quad \text{subject to} \quad \text{vol}(S) \in [v_l, v_h],$$

where the volume $\text{vol}(S) = \sum_{v_i \in S} d_i$ of a set is the sum of the degrees of its nodes.

The above can be shown to be NP-hard (Iyer et al., 2013) and exhibits strong connections with other classical formulations: it is a volume-balanced graph partitioning problem (Andreev and Racke, 2006) and can be used to minimize graph conductance (Chung and Graham, 1997) by scanning through solutions in different volume intervals and selecting the one whose cut-over-volume ratio is the smallest (this is how we test it in Section 2.4).

We employ the method described in Section 2.2.2 to derive a probabilistic loss function:

Corollary 2. Let the probabilities $p_1, \ldots, p_n$ giving rise to $\mathcal{D}$ be re-scaled such that $\sum_{v_i \in V} d_i p_i = \frac{v_l + v_h}{2}$ and, further, fix $\ell_{\text{cut}}(\mathcal{D}; G) \triangleq \sum_{v_i \in V} d_i p_i - 2\sum_{(v_i, v_j) \in E} p_i p_j w_{ij}$. Set $S^* \sim \mathcal{D}$ satisfies

$$\text{cut}\big(S^*\big) < \ell_{\text{cut}}(\mathcal{D}; G)/(1-t) \quad \text{and} \quad \text{vol}\big(S^*\big) \in [v_l, v_h],$$

with probability at least $t - 2\exp\big(-(v_h - v_l)^2 / \sum_i 2d_i^2\big)$.

The derived loss function ℓ_{cut} can be computed efficiently on a sparse graph, as its computational complexity is linear on the number of edges.

Decoding clusters. Retrieving a set that respects Corollary 2 can be done by sampling. Alternatively, the method described in Section 2.2.3 can guarantee that the identified cut is at most as small as the one certified by the probabilistic loss. In the latter case, the linear box constraint can be practically enforced by terminating before the volume constraint gets violated.

2.4 Empirical evaluation

We evaluate our approach in its ability to find large cliques and partitions of good conductance.

2.4.1 Methods

We refer to our network as Erdős' GNN, paying tribute to the pioneer of the probabilistic method that it is inspired from. Its architecture comprises of multiple layers of the Graph Isomorphism Network (GIN) (Xu et al., 2018) and a Graph Attention (GAT) (Veličković et al., 2017) layer. Furthermore, each convolution layer was equipped with skip connections, batch normalization and graph size normalization (Dwivedi et al., 2020). In addition to a graph, we gave our network access to a one-hot encoding of a randomly selected node, which encourages locality of solutions, allows for a trade-off between performance and efficiency (by rerunning the network with different samples), and helps the network break symmetries (Seo et al., 2019). Our network was trained with mini-batch gradient descent, using the Adam optimizer (Kingma and Ba, 2014) and was implemented on top of the pytorch geometric API (Fey and Lenssen, 2019).

Maximum clique. We compared against three neural networks, three discrete algorithms, and two integer-programming solvers: The neural approaches comprised of *RUN-CSP*, *Bomze GNN*, and *MS GNN*. The former is a SotA unsupervised network incorporating a reduction to independent set and a post-processing of invalid solutions with a greedy heuristic. The latter two, though identical in construction to Erdős' GNN, were trained based on standard smooth relaxations of the maximum clique problem with a flat 0.5-threshold discretization (Motzkin and Straus, 1965; Bomze, 1997). Since all these methods can produce multiple outputs for the same graph (by rerunning them with different random node attributes), we fix two time budgets for RUN-CSP and Erdős' GNN, that we refer to as "fast" and "accurate" and rerun them until the budget is met (excluding reduction costs). On the other hand, the Bomze and MS GNNs are rerun 25 times, since further repetitions did not yield relevant improvements. We considered the following algorithms: the standard *Greedy MIS Heur.* which greedily constructs a maximal independent set on the complement graph, *NX MIS approx.* (Boppana and Halldórsson, 1992), and *Toenshoff-Greedy* (Toenshoff et al., 2019). Finally, we formulated the maximum clique in integer form (Bomze et al., 1999) and solved it with *CBC* (johnjforrest et al., 2020) and *Gurobi* 9.0 (Gurobi Optimization, 2020), an open-source solver provided with Google's OR-Tools package and a SotA commercial solver. We should stress that our evaluation does not intend to establish SotA results (which would require a more exhaustive comparison), but aims to comparatively study the weaknesses and strengths of key unsupervised approaches.

Local partitioning. We compared against two neural networks and four discrete algorithms. To the extent of our knowledge, no neural approach for constrained partitioning exists in the literature. Akin to maximum clique, we built the *L1 GNN* and *L2 GNN* to be identical to Erdős' GNN and trained them based on standard smooth ℓ_1 and ℓ_2 relaxations of the cut combined with a volume penalty. On the other hand, a number of algorithms are known for finding small-volume sets of good conductance. We compare to well-known and advanced algorithms (Fountoulakis et al., 2018): *Pagerank-Nibble* (Andersen et al., 2006), Capacity Releasing Diffusion (*CRD*) (Wang et al., 2017), Max-flow Quotient-cut Improvement (*MQI*) (Lang and Rao, 2004) and *Simple-Local* (Veldt et al., 2016).

2.4.2 Data

Experiments for the maximum clique were conducted in the IMDB, COLLAB (Kersting et al., 2020; Yanardag and Vishwanathan, 2015) and TWITTER (Leskovec and Krevl, 2014) datasets, listed in terms of increasing graph size. Further experiments were done on graphs generated from the RB model (Xu et al., 2007), that has been specifically designed to generate challenging problem instances. We worked with three RB datasets: a training set containing graphs of up to 500 nodes (Toenshoff et al., 2019), a newly generated test set containing graphs of similar size, and a set of instances that are up to 3 times larger (Xu, 2007; Li et al., 2018; Toenshoff et al., 2019). On the other hand, to evaluate partitioning, we focused on the FACEBOOK (Traud et al., 2012), TWITTER, and SF-295 (Yan et al., 2008) datasets, with the first being a known difficult benchmark. More details can be found in the Appendix.

Evaluation. We used a 60-20-20 split between training, validation, and test for all datasets, except for the RB model data (details in paragraph above). Our baselines often require the reduction of maximum clique to independent set, which we have done when necessary. The reported time costs factor in the cost of reduction. During evaluation, for each graph, we sampled multiple inputs, obtained their solutions, and kept the best one. This was repeated for all neural approaches and local graph clustering algorithms. Solvers were run with multiple time budgets.

2.4.3 Results: maximum clique

Table 2.1 reports the test set approximation ratio, i.e., the ratio of each solution's cost over the optimal cost. For simple datasets, such as IMDB, most neural networks achieve similar performance and do not violate the problem constraints. On the other hand, the benefit of the probabilistic penalty method becomes clear on the more-challenging Twitter dataset, where training with smooth relaxation losses yields significantly worse results and constraint violation in at least 78% of the instances (see Appendix). Erdős' GNN always respected constraints. Our method was also competitive w.r.t. network RUN-CSP and the best solver, consistently giving better results when optimizing for speed ("fast"). The most accurate method overall was Gurobi, which impressively solved all instances perfectly given sufficient time. As observed, Gurobi has been heavily engineered to provide significant speed up w.r.t. CBC. Nevertheless, we should stress that both solvers scale poorly with the number of nodes and are not viable candidates for graphs with more than a few thousand nodes.

	IMDB	COLLAB	TWITTER
Erdős' GNN (fast)	1.000 (0.08 s/g)	0.982 ± 0.063 (0.10 s/g)	**0.924 ± 0.133 (0.17 s/g)**
Erdős' GNN (accurate)	1.000 (0.10 s/g)	0.990 ± 0.042 (0.15 s/g)	0.942 ± 0.111 (0.42 s/g)
RUN-CSP (fast)	0.823 ± 0.191 (0.11 s/g)	0.912 ± 0.188 (0.14 s/g)	0.909 ± 0.145 (0.21 s/g)
RUN-CSP (accurate)	0.957 ± 0.089 (0.12 s/g)	0.987 ± 0.074 (0.19 s/g)	0.987 ± 0.063 (0.39 s/g)
Bomze GNN	0.996 ± 0.016 (0.02 s/g)	*0.984 ± 0.053 (0.03 s/g)*	*0.785 ± 0.163 (0.07 s/g)*
MS GNN	0.995 ± 0.068 (0.03 s/g)	*0.938 ± 0.171 (0.03 s/g)*	*0.805 ± 0.108 (0.07 s/g)*
NX MIS approx.	0.950 ± 0.071 (0.01 s/g)	0.946 ± 0.078 (1.22 s/g)	0.849 ± 0.097 (0.44 s/g)
Greedy MIS Heur.	0.878 ± 0.174 (1e-3 s/g)	0.771 ± 0.291 (0.04 s/g)	0.500 ± 0.258 (0.05 s/g)
Toenshoff-Greedy	0.987 ± 0.050 (1e-3 s/g)	0.969 ± 0.087 (0.06 s/g)	**0.917 ± 0.126 (0.08 s/g)**
CBC (1s)	0.985 ± 0.121 (0.03 s/g)	0.658 ± 0.474 (0.49 s/g)	0.107 ± 0.309 (1.48 s/g)
CBC (5s)	1.000 (0.03 s/g)	0.841 ± 0.365 (1.11 s/g)	0.198 ± 0.399 (4.77 s/g)
Gurobi 9.0 (0.1s)	**1.000 (1e-3 s/g)**	0.982 ± 0.101 (0.05 s/g)	0.803 ± 0.258 (0.21 s/g)
Gurobi 9.0 (0.5s)	1.000 (1e-3 s/g)	0.997 ± 0.035 (0.06 s/g)	0.996 ± 0.019 (0.34 s/g)
Gurobi 9.0 (1s)	1.000 (1e-3 s/g)	0.999 ± 0.015 (0.06 s/g)	**1.000 (0.34 s/g)**
Gurobi 9.0 (5s)	1.000 (1e-3 s/g)	**1.000 (0.06 s/g)**	1.000 (0.35 s/g)

Table 2.1: Test set approximation ratios for all methods on real-world datasets. For solvers, time budgets are listed next to the name. Pareto-optimal solutions are indicated in bold, whereas italics indicate constraint violation (we report the results only for correctly solved instances).

Table 2.2 tests the best methods on hard instances. We only provide the results for Toenshoff-Greedy, RUN-CSP, and Gurobi, as the other baselines did not yield meaningful results. Erdős' GNN can be seen to be better than RUN-CSP in the training and test set and worse for larger, out of distribution, instances. However, both neural approaches fall behind the greedy algorithm and Gurobi, especially when optimizing for quality. The performance gap is pronounced for small instances but drops significantly for larger graphs, due to Gurobi's high computational complexity. It is also interesting to observe that the neural approaches do better on the training set than on the test set. Since both neural methods are completely unsupervised, the training set performance can be taken at face value (the methods never saw any labels). Nevertheless, the results also show that both methods partially overfit the training distribution. The main weakness of Erdős' GNN is that its performance degrades when testing it in larger problem instances. Nevertheless, it is encouraging to observe that even on graphs of at most 1500 nodes, both our "fast" method and RUN-CSP surpass Gurobi when given the same time-budget. We hypothesize that this phenomenon will be more pronounced with larger graphs.

2.4.4 Results: local graph partitioning

The results of all methods and datasets are presented in Table 2.3. To compare fairly with previous works, we evaluate partitioning quality based on the measure of local conductance, $\phi(S) = \text{cut}(S)/\text{vol}(S)$, even though our method only indirectly optimizes conductance. Nev-

	Training set	Test set	Large Instances
Erdős' GNN (fast)	0.899 ± 0.064 (0.27 s/g)	0.788 ± 0.065 (0.23 s/g)	0.708 ± 0.027 (1.58 s/g)
Erdős' GNN (accurate)	0.915 ± 0.060 (0.53 s/g)	0.799 ± 0.067 (0.46 s/g)	0.735 ± 0.021 (6.68 s/g)
RUN-CSP (fast)	0.833 ± 0.079 (0.27 s/g)	0.738 ± 0.067 (0.23 s/g)	0.771 ± 0.032 (1.84 s/g)
RUN-CSP (accurate)	0.892 ± 0.064 (0.51 s/g)	0.789 ± 0.053 (0.47 s/g)	0.804 ± 0.024 (5.46 s/g)
Toenshoff-Greedy	**0.924 ± 0.060 (0.02 s/g)**	0.816 ± 0.064 (0.02 s/g)	**0.829 ± 0.027 (0.35 s/g)**
Gurobi 9.0 (0.1s)	0.889 ± 0.121 (0.18 s/g)	**0.795 ± 0.118 (0.16 s/g)**	0.697 ± 0.033 (1.17 s/g)
Gurobi 9.0 (0.5s)	**0.962 ± 0.076 (0.34 s/g)**	**0.855 ± 0.083 (0.31 s/g)**	0.697 ± 0.033 (1.54 s/g)
Gurobi 9.0 (1.0s)	**0.980 ± 0.054 (0.45 s/g)**	**0.872 ± 0.070 (0.40 s/g)**	0.705 ± 0.039 (2.05 s/g)
Gurobi 9.0 (5.0s)	**0.998 ± 0.010 (0.76 s/g)**	**0.884 ± 0.062 (0.68 s/g)**	0.790 ± 0.285 (6.01 s/g)
Gurobi 9.0 (20.0s)	**0.999 ± 0.003 (1.04 s/g)**	**0.885 ± 0.063 (0.96 s/g)**	0.807 ± 0.134 (21.24 s/g)

Table 2.2: Hard maximum clique instances (RB). We report the approximation ratio (bigger is better) in the training and test set, whereas the rightmost column focuses on a different distribution consisting of graphs of different sizes. Execution time is measured in sec. per graph (s/g). Pareto-optimal solutions are in bold.

ertheless, Erdős' GNN outperforms all previous algorithms by a considerable margin. We would like to stress that this result is not due to poor usage of previous methods: we rely on a well-known implementation (Fountoulakis et al., 2018) and select the parameters of all non-neural baselines by grid-search on a held-out validation set. We also do not report performance when a method (Pagerank-Nibble) returns the full graph as a solution (Wang et al., 2017).

It is also interesting to observe that, whereas all neural approaches perform well, GNN trained with a probabilistic loss attains better conductance across all datasets. We remind the reader that all three GNNs feature identical architectures and that the L1 and L2 loss functions are smooth relaxations that are heavily utilized in partitioning problems (Bresson et al., 2013). Furthermore, due to its high computational complexity and the extra overhead that is incurred when constructing the problem instances for large graphs, Gurobi performed poorly in all but the smallest graphs.We argue that the superior solution quality of Erdős' GNN serves as evidence for the benefit of our unsupervised framework.

2.4.5 Visual demonstration

Figure 2.2 provides a visual demonstration of the input and output of Erdős' GNN in a simple instance of the maximum clique problem.

We would like to make two observations. The first has to do with the role of the starting seed in the probability assignment produced by the network. In the maximum clique problem, we did not require the starting seed to be included in the solutions. This allowed the network to flexibly detect maximum cliques within its receptive field without being overly constrained by the random seed selection. This is illustrated in the example provided in the figure, where the seed is located inside a smaller clique and yet the network is able to produce probabilities

	SF-295	FACEBOOK	TWITTER
Erdős' GNN	**0.124 ± 0.001 (0.22 s/g)**	**0.156 ± 0.026 (289.28 s/g)**	**0.292 ± 0.009 (6.17 s/g)**
L1 GNN	0.188 ± 0.045 (0.02 s/g)	0.571 ± 0.191 (13.83 s/g)	**0.318 ± 0.077 (0.53 s/g)**
L2 GNN	**0.149 ± 0.038 (0.02 s/g)**	**0.305 ± 0.082 (13.83 s/g)**	0.388 ± 0.074 (0.53 s/g)
Pagerank-Nibble	0.375 ± 0.001 (1.48 s/g)	N/A	0.603 ± 0.005 (20.62 s/g)
CRD	0.364 ± 0.001 (0.03 s/g)	0.301 ± 0.097 (596.46 s/g)	0.502 ± 0.020 (20.35 s/g)
MQI	0.659 ± 0.000 (0.03 s/g)	0.935 ± 0.024 (408.52 s/g)	0.887 ± 0.007 (0.71 s/g)
Simple-Local	0.650 ± 0.024 (0.05 s/g)	0.955 ± 0.019 (404.67 s/g)	0.895 ± 0.008 (0.84 s/g)
Gurobi (10s)	**0.105 ± 0.000 (0.16 s/g)**	0.961 ± 0.010 (1787.79 s/g)	0.535 ± 0.006 (52.98 s/g)

Table 2.3: Cluster conductance on the test set (smaller is better) and execution time measured in sec. per graph. Pareto-optimal solutions are in bold.

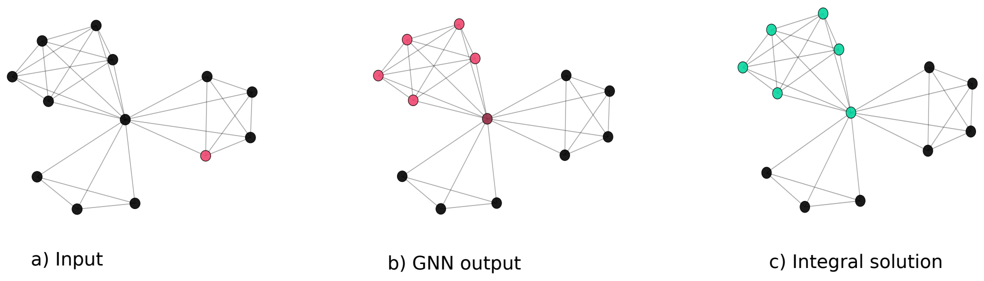


Figure 2.2: Illustration of our approach in a toy instance of the maximum clique problem from the IMDB dataset. a) A random node is selected to act as a 'seed'. b) Erdős' GNN outputs a probability distribution over the nodes (color intensity represents the probability magnitude) by exploring the graph in the vicinity of the seed. c) A set is sequentially decoded by starting from the node whose probability is the largest and iterating with the method of conditional expectation. The identified solution is guaranteed to obey the problem constraints, i.e., to be a clique.

that focus on the largest clique. On the other hand, in the local graph partitioning problem we forced the seed to always lie in the identified solution—this was done to ensure a fair comparison with previous methods. Our second observation has to do with the sequential decoding process. It is encouraging to notice that, even though the central hub node has a considerably lower probability than the rest of the nodes in the maximum clique, the method of conditional expectation was able to reliably decode the full maximum clique.

2.4.6 Maximum clique problem: ablations

The following experiments provide evidence that both the learning and decoding phases of our framework are important in obtaining valid cliques of large size.

Constraint violation

Table 2.4 reports the percentage of instances in which the clique constraint was violated in our experiments. Neural baselines optimized according to penalized continuous relaxations struggle to detect cliques in the COLLAB and TWITTER datasets, whereas Erdős' GNN always respected the constraint.

	IMDB	COLLAB	TWITTER	RB (all datasets)
Erdős' GNN (fast)	**0%**	**0%**	**0%**	**0%**
Erdős' GNN (accurate)	**0%**	**0%**	**0%**	**0%**
Bomze GNN	**0%**	11.8%	78.1%	–
MS GNN	1%	15.1%	84.7%	–

Table 2.4: Percentage of test instances where the clique constraint was violated.

Thus, decoding solutions by the method of conditional expectation is crucial to ensure that the clique constraint is always satisfied.

Importance of learning

We also tested the efficacy of the learned probability distributions produced by our GNN on the Twitter dataset. We sampled multiple random seeds and produced the corresponding probability assignments by feeding the inputs to the GNN. These were then decoded with the method of conditional expectation and the best solution was kept. To measure the contribution of the GNN, we compared to random uniform probability assignments on the nodes. In that case, instead of multiple random seeds, we had the same number of multiple random uniform probability assignments. Again, these were decoded with the method of conditional expectation and the best solution was kept. The results of the experiment can be found in Table 2.5.

	Erdős' GNN	$U \sim [0,1]$
1 sample	0.821 ± 0.222	0.513 ± 0.266
3 samples	0.875 ± 0.170	0.694 ± 0.210
5 samples	0.905 ± 0.139	0.760 ± 0.172

Table 2.5: Approximation ratios with sequential decoding using the method of conditional expectation on the twitter dataset. The second column represents decoding with the probabilities produced by the GNN. The third column shows the results achieved by decoding random uniform probability assignments on the nodes.

As observed, the cliques identified by the trained GNN were significantly larger than those obtained when decoding a clique from a random probability assignment.

2.4.7 Local graph partitioning: additional results

We also attempted to find sets of small conductance using Gurobi. To ensure a fair comparison, we mimicked the setting of Erdős' GNN and re-run the solver with three different time-budgets, making sure that the largest budget exceeded our method's running time by approximately one order of magnitude. We used the following integer-programming formulation of the constrained graph partitioning problem:

$$\min_{x_1,\ldots,x_n \in \{0,1\}} \sum_{(v_i,v_j)\in E} (x_i - x_j)^2 \tag{2.7}$$

$$\text{subject to} \quad \left(1-\frac{1}{4}\right)\text{vol} \le \sum_{v_i\in V} x_i d_i \le \left(1+\frac{1}{4}\right)\text{vol} \quad \text{and} \quad x_s = 1.$$

Above, vol is a target volume and s is the index of the seed node (see explanation in Section A.2.3). Each binary variable x_i is used to indicate membership in the solution set. In order to encourage local solutions on a global solver like Gurobi, the generated target volumes were set to lie in an interval that is attainable within a fixed receptive field (identically to the neural baselines). Additionally, the seed node v_s was also required to be included in the solution. The above choices are consistent with the neural baselines and the local graph partitioning setting.

The results are shown in Table 2.6. Due to its high computational complexity, Gurobi performed poorly in all but the smallest instances. In the FACEBOOK dataset, which contains graphs of 7k nodes on average, Erdős' GNN was impressively able to find sets of more than 6× smaller conductance, while also being 6× faster.

	SF-295	FACEBOOK	TWITTER
Gurobi (0.1s)	0.107 ± 0.000 (0.16 s/g)	0.972 ± 0.000 (799.508 s/g)	0.617 ± 0.012 (3.88 s/g)
Gurobi (1s)	0.106 ± 0.000 (0.16 s/g)	0.972 ± 0.000 (893.907 s/g)	0.544 ± 0.007 (12.41 s/g)
Gurobi (10s)	**0.105 ± 0.000 (0.16 s/g)**	0.961 ± 0.010 (1787.79 s/g)	0.535 ± 0.006 (52.98 s/g)
Erdős' GNN	0.124 ± 0.001 (0.22 s/g)	**0.156 ± 0.026 (289.28 s/g)**	**0.292 ± 0.009 (6.17 s/g)**

Table 2.6: Average conductance of sets identified by Gurobi and Erdős' GNN (these results are supplementary to those of Table 2.3).

It should be noted that the time budget allowed for Gurobi only pertains to the *optimization time* spent (for every seed). There are additional costs in constructing the problem instances and their constraints for each graph. These costs become particularly pronounced in larger graphs, where setting up the problem instance takes more time than the allocated optimization budget. We report the total time cost in seconds per graph (s/g).

2.5 Learning with the probabilistic method: extended discussion

In the previous sections, we presented fundamental tools of the probabilistic method that can be used for unsupervised training of neural networks and showed that they can lead to empirically successful neural network pipelines for combinatorial optimization. Since the publication of that work, the method has been utilized in several follow-up works to solve numerous combinatorial optimization problems (McCarty et al., 2021; Wang et al., 2022; Lin et al., 2022; Min et al., 2022; Dai et al., 2020). This has also brought to the surface some of the common challenges that are faced when using the probabilistic method in a neural network context. This motivates the topics discussed in this section, which aim to elucidate some of the subtle details behind this probabilistic approach, some of the key challenges that the framework is facing, and some directions that can improve over the recipe that was presented in the original publication.

2.5.1 The unconstrained case

First, we begin by providing a more detailed discussion for the definition of the loss function in the unconstrained case in section 2.2.2. Recall that we are seeking a formal characterization for the quality of the solution generated by the neural network. In the unconstrained case, a probabilistic guarantee would require that with some probability t, a solution of bounded cost is found, i.e.,

$$P(f(S;G) < \ell(\mathcal{D};G)) > t, \tag{2.8}$$

for some learned distribution $\mathcal{D} = g_\theta(G)$. Our goal will be to find a differentiable loss function $\ell(\mathcal{D};G)$ that satisfies this property. This is achieved in 2.3. More concretely, if we assume that f is non-negative, from Markov's inequality we have

$$P\left(f(S;G) \geq \ell(\mathcal{D};G)\right) \leq \frac{\mathbb{E}_{S\sim\mathcal{D}}[f(S;G)]}{\ell(\mathcal{D};G)}. \tag{2.9}$$

For convenience, we will suppress the subscript in $\mathbb{E}_{S\sim\mathcal{D}}$ notation. But it should be clear that throughout the chapter that the expectation is taken over the parametrized distribution of sets for a given graph G. Then, from 2.8

$$1 - P(f(S;G) \geq \ell(\mathcal{D};G)) > t. \tag{2.10}$$

Obviously, if the above inequality holds for the upper bound $\mathbb{E}[f(S;G)]/\ell(\mathcal{D};G)$, then it also holds for $P(f(S;G) \geq \ell(\mathcal{D};G))$. We may replace the probability with the upper bound to obtain

$$1 - \frac{\mathbb{E}[f(S;G)]}{\ell(\mathcal{D};G)} > t,$$

and consequently

$$\ell(\mathcal{D}; G) > \frac{\mathbb{E}[f(S;G)]}{1-t}, \quad t \in [0,1).$$

Now, let us consider the case where $\ell(\mathcal{D}; G) = \epsilon$. This implies

$$\epsilon(1-t) > \mathbb{E}[f(S;G)].$$

The fact that f is nonnegative and the inequality above imply that there exists a set S with cost at most ϵ. This leads us to the definition presented in 2.3.

2.5.2 The minima of the probabilistic penalty function

An important consideration has to do with the minima of the probabilistic penalty loss. Recall that the minimum of a given problem is given by $S^* = \min_{S \in \Omega} f(S;G)$, and that the probabilistic penalty loss is given by

$$\ell(\mathcal{D}; G) = \mathbb{E}\left[f(S;G)\right] + \beta P(S \notin \Omega).$$

Theorem 3. The minima of the probabilistic penalty loss are the minima of the constrained optimization problem:

$$\min_{\mathcal{D}} \ell(\mathcal{D}; G) = \min_{S \in \Omega} f(S;G).$$

See proof in Appendix A.1.4.

Of course, the fact that the minima of the functions match does not provide any guarantees for the optimization landscape itself or the performance of the model. However, it does guarantee that our continuous relaxation of the problem does not introduce spurious minima and that the loss “faithfully” represents the problem.

2.6 Modelling constraints: case studies

In this section, we will briefly go over a list of commonly encountered combinatorial problems and work out the corresponding derivations for their loss functions. Some of the examples follow naturally from the derivations that have already been provided in the case studies on maximum clique and partitioning. Here we make these connections explicit. The list of problems that are covered here is by no means exhaustive, but the goal of the section is to serve as a demonstration for the applicability of the method to some of the most well-known combinatorial problems and to make the methodology completely transparent. These demonstrations will also serve as a motivation to discuss the central challenges of the framework later in this chapter.

2.6.1 Constrained optimization: cliques, covers, and more

Maximum independent set. In the simple unweighted case of the maximum independent set problem, we are given a graph $G = (V, E)$ and we are looking for the largest set S that contains nodes that are not connected to each other, i.e., $i \not\sim j$ for all $i, j \in S$. The problem is known to be NP-Hard. This can be written as the following optimization problem:

$$\max_S |S| \quad \text{subject to} \quad S \in \Omega_{\text{indep}}.$$

Recall that the probabilistic penalty loss is defined as

$$\ell(\mathcal{D}; G) \triangleq \mathbb{E}[f(S; G)] + \beta P(S \notin \Omega).$$

Let $\gamma > \max_{S \in G} |S|$. Then the unconstrained cost will be $f(S;G) = \gamma - |S|$. The probability of constraint violation (i.e., the probability that the set is not independent), is equal to the probability that sets S will contain at least one edge. Formally,

$$\begin{aligned} P(S \notin \Omega_{\text{indep}}) &= P\left(\sum_{v_i, v_j \in S} \mathbf{1}_{\{(v_i, v_j) \in E\}} \geq 1 \right) \\ &\leq \mathbb{E}\left[\sum_{v_i, v_j \in S} x_i x_j w_{ij} \right] && \text{(Markov's Inequality)} \\ &= \mathbb{E}\left[w(S) \right] \\ &= \sum_{v_i, v_j \in S} p_i p_j (w_{ij}). \end{aligned}$$

Here, $\mathbb{E}\left[\bar{w}(S)\right]$ denotes the "expected weight" of the graph, i.e., the expected number of edges contained by sets S. This leads to the following loss

$$\ell_{\text{indep}}(\mathcal{D}; G) \triangleq \gamma - \sum_{i \in V} p_i + \beta \sum_{v_i, v_j \in S} p_i p_j (w_{ij}).$$

Observe how the probability constraint violation resembles the one derived for the maximum clique problem. As we have pointed out earlier, there exists a straightforward reduction from maximum clique to the independent set problem: the maximum clique in a graph G is equal to the maximum independent set in the complement $\bar{G}$. This means that we can rewrite the clique constraint in A.6 with respect to the complement graph,

$$\begin{aligned}
P(S \notin \Omega_{\text{clique}}) &= P\left(\sum_{v_i,v_j \in S} \mathbf{1}_{\{(v_i,v_j)\notin E\}} \geq 1\right) \\
&\leq \mathbb{E}\left[\sum_{v_i,v_j \in S} x_i x_j \bar{w}_{ij}\right] && \text{(Markov's Inequality)} \\
&= \mathbb{E}\left[\bar{w}(S)\right] \\
&= \sum_{v_i,v_j \in S} p_i p_j \bar{w}_{ij} \\
&= \sum_{v_i,v_j \in S} p_i p_j (1 - w_{ij}).
\end{aligned}$$

For simple unweighted graphs we have $w_{ij} = 1$ if $v_i, v_j \in E$ and $w_{ij} = 0$ otherwise.

Maximum cut. In the maximum cut problem, given a graph $G = (V, E)$, we are tasked with finding the set of nodes S with the maximum cut, i.e., with the maximum number of edges in E that have exactly one endpoint in S and one in $V \setminus S$. Recall the expected value of the cut from A.11,

$$E[\text{cut}(S)] = \sum_{v_i \in V} d_i p_i - 2 \sum_{(v_i,v_j)\in E} p_i p_j w_{ij}.$$

Then we may use $\gamma = |E|$ to obtain the cut loss:

$$\ell_{\text{maxcut}}(\mathcal{D}; G) = \gamma - \mathbb{E}[\text{cut}(S)] = |E| - \sum_{v_i \in V} d_i p_i - 2 \sum_{(v_i,v_j)\in E} p_i p_j w_{ij}.$$

Minimum vertex cover. In the minimum vertex cover, we are tasked with finding the smallest set S in a graph $G = (V, E)$ such that for all edges $(v_i, v_j) \in E$ in the graph, we have that either $v_i \in S$ or $v_j \in S$. The corresponding optimization problem can be written as:

$$\min_S |S| \quad \text{subject to} \quad S \in \Omega_{\text{cover}}.$$

We may calculate the probability of constraint violation,

$$\begin{aligned}
P(S \notin \Omega_{\text{cover}}) &= P\left(\sum_{(v_i,v_j)\in E} \mathbf{1}_{\{v_i \notin S \text{ and } v_j \notin S\}} \geq 1\right) \\
&\leq \mathbb{E}\left[\sum_{(v_i,v_j)\in E} (1 - x_i)(1 - x_j) w_{ij}\right] \\
&= \sum_{(v_i,v_j)\in E} (1 - p_i)(1 - p_j) w_{ij}
\end{aligned}$$

Therefore, the derived loss for the minimum vertex cover problem will be

$$\ell_{\text{mincover}}(\mathcal{D};G) \triangleq \sum_{i\in V} p_i + \beta \sum_{v_i,v_j\in S} (1-p_i)(1-p_j)w_{ij}.$$

It is worth pointing out that the vertex cover is also related to the independent set and clique problems through a simple reduction. This can be observed in the formal similarities between the constraints as well. More concretely, for any independent set S of a graph G, its complement $\bar{S}$ forms a vertex cover.

Minimum Dominating Set. The minimum dominating set problem is closely related to the vertex cover problem. We are asked to find the smallest set S in a graph $G=(V,E)$ such that every node in the graph is either part of S or adjacent to some vertex in S. Again, we write the optimization problem as

$$\min_S |S| \quad \text{subject to} \quad S\in\Omega_{\text{domset}}.$$

To calculate the probability of constraint violation, let $\mathcal{N}_i^+ = \{v_j : v_j\in V, (v_j,v_i)\in E\}\cup\{v_i\}$.

$$\begin{aligned} P(S\notin\Omega_{\text{domset}}) &= P\left(\sum_{(v_i)\in V}\bigcap_{j\in\mathcal{N}_i^+} \mathbf{1}_{v_j\notin S}\geq 1\right) \\ &\leq \mathbb{E}\left[\sum_{v_i\in V}\prod_{j\in\mathcal{N}_i^+}(1-x_j)\right] \\ &= \sum_{(v_i)\in V}\prod_{j\in\mathcal{N}_i^+}(1-p_j). \end{aligned}$$

This leads to the derived loss for the minimum dominating set problem,

$$\ell_{\text{mindomset}}(\mathcal{D};G) \triangleq \sum_{i\in V} p_i + \beta\sum_{v_i\in V}\prod_{j\in\mathcal{N}_i^+}(1-p_j).$$

Again, the apparent formal similarity to the vertex cover problem is not an accident. It is known that for a connected graph, any vertex cover is also a dominating set. On the other hand the converse is not always true.

The case of multiple constraints In real world problems, it is often the case that multiple types of constraints have to be imposed simultaneously. Let $\Omega = \Omega_1\cup\Omega_2\cup\cdots\cup\Omega_k$ be the feasible set for a collection of k different constraints. Furthermore, assume that the each constraint admits a differentiable upper bound, i.e., $P(S\notin\Omega_i)$ can be differentiable for all $i=1,2,\ldots,k$. Then the corresponding probabilistic penalty loss is written as

$$\ell(\mathcal{D};G) = \mathbb{E}[f(S;G)] + \beta P(S\notin\bigcup_{i=1}^{n}\Omega_i).$$

It is straightforward to apply the union bound and obtain the following loss

$$\ell(\mathcal{D};G) = \mathbb{E}[f(S;G)] + \beta \sum_{i=1}^{n} P(S \notin \Omega_i). \tag{2.11}$$

2.6.2 Constraint satisfaction and boolean satisfiability

In the most general setting, we are dealing with problems defined over a finite collection of variables, with a given set of constraints over these variables, and our goal is to find an assignment of values to the variables such that all the constraints are satisfied. Formally, we are given a problem instance $\mathrm{I} = (\mathbf{X}, c_1, \ldots, c_m)$ which consists of variables $\mathbf{X} = (X_1, X_2, \ldots, X_n) \in \mathbb{X}^n$ that are defined over a discrete finite domain, and constraints $c_1, c_2, \ldots, c_m$, with $c_j : \mathbb{X}^n \to \{\mathsf{True}, \mathsf{False}\}$. We say that a problem instance is satisfiable by an assignment of variables $\mathbf{X}$ if $\bigcap_{j=1}^{m} c_j(\mathbf{X}) = \mathsf{True}$. The variables involved in a constraint c_j will be denoted as $\mathrm{vbl}(c_j)$, and are typically a subset of $\mathbf{X}$. It will be instructive to keep CNF-SAT as a working example as it will help us concretely the central points of this section. Boolean satisfiability is a central problem in computer science that has served as the foundation of the theory of computational complexity while also being intimately connected to important applications like software verification and computer chip design (Biere et al., 2009).

In its standard form, we are given a formula composed of a set of m clauses and a set of n Boolean variables. More specifically, each clause contains *literals*, i.e., variables v_i or their negation $\neg v_i$. A clause is said to be *satisfied* if it evaluates to TRUE. Each clause imposes logical constraints between literals, and the formula is said to be satisfiable if there exists an assignment of values to the variables that satisfies all the clauses.

For the rest of this discussion, we will be focusing on one of the most well-known versions of Boolean satisfiability, the CNF-SAT problem. In CNF-SAT, the formula is presented in *conjunctive normal form*. That means that each clause in the formula consists of a disjunction of literals (logical OR) and the formula consists of a conjunction (logical AND) of such clauses. Then, a formula is said to be *satisfiable* if there exists an assignment of values to the variables that makes all clauses simultaneously true. For example, the formula

$$(X_1 \vee X_2) \wedge (\neg X_1 \vee X_3)$$

admits multiple assignments that satisfy the formula, e.g., $X_1 = \mathsf{True}, X_2 = \mathsf{False}, X_3 = \mathsf{True}$. The Boolean satisfiability problem can be viewed both from a constraint satisfaction and an optimization lens. First, we replace logical variables X_i with discrete variables $x_i \in \{0,1\}$. Let $\mathcal{C}$ be the set of clauses in a given formula and let $\mathrm{vbl}^+(c_i), \mathrm{vbl}^-(c_i)$ be the sets of variables in c_i that appear as X_i and as their negations $\neg X_i$ respectively. We may then describe the number of unsatisfied clauses by an assignment to the variables $x_1, x_2, \ldots, x_n$ using a function f, defined as

$$f_{\text{unsat}}(x_1, x_2, \dots, x_n) = \sum_{c_i \in \mathcal{C}} \left(\prod_{x_j \in \text{vbl}^+(c_i)} (1 - x_j) \prod_{x_j \in \text{vbl}^-(c_i)} x_j \right).$$

Note that defining f using the number of unsatisfied clauses is a matter of convenience and helps simplify some expressions, as it will become apparent soon.

We begin with the optimization setting by looking at MAX-SAT. In this version of the problem, regardless of the satisfiability of the formula as a whole, we are looking to maximize the number of clauses that are true. The problem is known to be NP-Hard, even in the case of the 2-SAT (Garey et al., 1974). Therefore, the problem boils down to maximizing f and we can apply a similar probabilistic argument as we did in the unconstrained case to obtain a loss function. Namely, we have

$$\ell_{\text{maxSAT}}(\mathcal{D}; \text{I}) = \mathbb{E}[f_{\text{unsat}}(x_1, x_2, \dots, x_n)],$$

with a learned distribution $\mathcal{D}$ over possible assignments $\{0,1\}^n$. Similar to our previous examples, it is instantiated through probabilities p_i corresponding to each independent x_i variable. Therefore, from the linearity of expectation, we obtain

$$\mathbb{E}[f_{\text{unsat}}(x_1, x_2, \dots, x_n)] = \sum_{c_i \in \mathcal{C}} \left(\prod_{x_j \in \text{vbl}^+(c_i)} (1 - p_j) \prod_{x_j \in \text{vbl}^-(c_i)} p_j \right).$$

Boolean satisfiability may also be described as a constraint satisfaction problem. In that case, we are looking to decide whether the formula is satisfiable, i.e., to determine whether there exist assignments $S \in \{0,1\}^n$ that satisfy the entire formula (i.e., satisfy all the clauses jointly). This gives rise to classic problems like k-SAT, where each formula is composed of clauses that contain k literals.

We may again employ the probabilistic method in this scenario in order to tackle formula satisfiability. More concretely, if we show that the probability that all the clauses can be satisfied is positive, i.e., $P(S \in \Omega_{\text{SAT}}) > 0$, we have shown that the formula is satisfiable.

To do so, may consider only the constraint violation term in 2.4, i.e., we write

$$\ell_{\text{SAT}}(\mathcal{D}; \text{I}) \triangleq P(S \notin \Omega_{\text{SAT}}).$$

From this, it is clear that if $\ell_{\text{SAT}}(\mathcal{D}; \text{I}) < 1$, then the formula is satisfiable. Now, all we need is to find a differentiable upper bound for $P(S \notin \Omega_{\text{SAT}})$. Leveraging Markov's inequality yet again, we obtain

$$\begin{aligned}
P(S \notin \Omega_{\mathrm{SAT}}) &= P(f_{\mathrm{unsat}}(x_1, x_2, \dots, x_n) \geq 1) \\
&\leq \mathbb{E}[f_{\mathrm{unsat}}(x_1, x_2, \dots, x_n)] \\
&= \sum_{c_i \in \mathcal{C}} \left(\prod_{x_j \in \mathrm{vbl}^+(c_i)} (1 - p_j) \prod_{x_j \in \mathrm{vbl}^-(c_i)} p_j \right).
\end{aligned}$$

As we can see the losses obtained for the unconstrained optimization version and the constraint satisfaction version are the same up to rescaling. This is due to the decomposability of the problem in terms of clause events. More concretely, let $b_i = \{S : c_i(S) = \mathrm{FALSE}\}$, i.e., b_i is the set of assignments that lead to the failure of clause i (clause i being unsatisfied). Each clause c_i can be then seen to define a feasible region Ω_i and $b_i = \bar{\Omega}_i$ its complement. This means we may calculate the probability as $p(S \in b_i) = \prod_{x_j \in \mathrm{vbl}^+(c_i)} (1 - p_j) \prod_{x_j \in \mathrm{vbl}^-(c_i)} p_j$. This follows easily from the observation that a CNF clause fails only if all literals in that clause simultaneously evaluate to false.

Indeed, we can use the union bound strategy from 2.11 and bound the probability of satisfiability

$$\begin{aligned}
P(S \notin \Omega_{\mathrm{SAT}}) &= P\left(\bigcup_{c_i \in \mathcal{C}} S \in b_i \right) \\
&\leq \sum_{c_i \in \mathcal{C}} P(S \in b_i) \\
&= \sum_{c_i \in \mathcal{C}} \left(\prod_{x_j \in \mathrm{vbl}^+(c_i)} (1 - p_j) \prod_{x_j \in \mathrm{vbl}^-(c_i)} p_j \right).
\end{aligned}$$

Again, we arrive at the same expression. This is a direct consequence of the fact that when we use Markov's inequality, we leverage the linearity of expectation to calculate the expected values of sums. Those sums can also be seen to emerge by taking the union bound over the union of events over edges (or clauses in the case of SAT).

2.7 Challenges of learning with the probabilistic method

The previous sections have shown how the probabilistic method may be used to successfully model a plethora of combinatorial problems. Naturally, there are properties that make certain problems more amenable to the probabilistic method approach and others more challenging to deal with. In this section, we will provide an in depth discussion of the most challenging aspects of applying the probabilistic method in combinatorial problems along with how they could potentially be addressed. First, we summarize the 3 most significant challenges:

- **Challenge 1.** The method, as it has been proposed, relies on rather simple bounds which may not adequately capture the problem structure.

- **Challenge 2.** The method requires explicit access to the objective, i.e., it cannot work in a setting where the objective of the problem is provided as a black box.

- **Challenge 3.** The method requires the derivation of an efficiently computable and differentiable expectation for the objective, and a differentiable upper bound or expectation for the constraint violation. This becomes harder to achieve with the product measure when the constraints or the objective are more complicated (e.g., nonlinear) functions.

Note that there is an overlap between the challenges. For example, the requirement for a differentiable expectation for the cost and a differentiable bound for the constraint violation (challenge 3) cannot be satisfied in the case of black box objectives and constraints (challenge 2). On the other hand these challenges are presented separately as it is not clear whether dealing with challenge 3 necessarily implies a solution to challenge 2. One could plausibly develop techniques for analytically computing differentiable bounds for more complicated constraints by using more sophisticated probabilistic tools while keeping the product measure and ignoring the black-box issue altogether.

2.7.1 Challenge 1: On the limitations of simple bounds

We will illustrate this challenge by focusing on the problem of boolean satisfiability, where some of the limitations of using bounds like Markov's inequality and the union bound can become immediately apparent. To make this more concrete, consider the following example. Let $p_i = \frac{1}{2}$ be the probability that the variable v_i is True for all v_i and consider the problem of 3-SAT, where each clause contains a disjunction of precisely 3 literals. Then, for any clause c_i, $P(S \in b_i) = \frac{1}{2^3}$. In that case, the union bound implies

$$P(S \notin \Omega_{\mathrm{SAT}}) \leq \frac{m}{2^3}.$$

This bound depends directly on the size of the formula (number of clauses). Clearly, as the size of the formulae gets larger, if we were to use this as a loss function, the neural network would have to increasingly get closer to the correct assignment in order to obtain a satisfiability certificate.

Another issue that is brought by the use of simple bounds has to do with the dependencies between events. For example, the union bound will be tight only in the case of independent events. However, this is clearly not the case for any non-trivial formula, as the satisfiability of a clause can provide information about the satisfiability of neighboring clauses due to the fact that variables may occur in several clauses. We will make this observation precise in the following section. It will motivate a different probabilistic approach that will be able to deal with both scalability and dependency issues.

2.7.2 Challenges 2 & 3: Black box functions and complex constraints

It may be apparent to some readers that the challenges that were listed earlier are more closely connected than what they might appear. For example, the use of a product measure commits us to certain types of bounds (challenge 1) which may limit our ability to model particular kinds of constraints (challenge 3). As another example, if we could handle the black-box setting (challenge 2), then that could potentially provide (partial) solutions to the rest of the challenges. To highlight these connections we will be discussing the remaining two challenges jointly.

To model an arbitrary combinatorial problem, a starting point is to consider its integer programming formulation, if available. For example, consider the integer program for the Travelling Salesperson Problem (TSP)(Dantzig et al., 1954) on a complete graph with distances w_{ij} and variables x_{ij} per edge:

$$\min \sum_{(v_i,v_j)\in E} x_{ij} w_{ij}, \tag{2.12}$$

$$\text{subject to } \sum_i x_{ij} = \sum_j x_{ij} = 1, \quad \text{for all } i, j, \tag{2.13}$$

$$\sum_{i\in T} \sum_{j\in T} x_{ij} \le |T| - 1 \quad \text{for all } S \subset V \text{ and } S \neq \emptyset, \quad x_{ij} \in \{0,1\}. \tag{2.14}$$

First, we consider the cost function and the equality constraints in 2.13. Since the cost function is linear, its expectation for the probabilistic penalty loss can be easily derived from the linearity of expectation. The way to compute the probability of constraint violation for the equality constraints in 2.13 with a differentiable expression is not trivial but may be achieved through a more elaborate argument. The inequality in 2.14 requires checking exponentially many sets to enforce the subtour constraint. Various heuristics may be considered in order to effectively deal with this constraint, e.g., a parametric connectivity constraint was recently utilized by Gaile et al. (2023) to facilitate an unsupervised learning pipeline for the TSP. In the case of the probabilistic method, it is nontrivial to determine how one could obtain a valid upper bound on the probability of constraint violation through heuristics. Furthermore, if we commit to some problem-specific heuristic, then clearly we would have to repeat that process on a case-by-case basis which harms the overall applicability of our method.

Here, the central vulnerability of the approach we described so far becomes transparent. The expectations of objective functions and the probabilities of constraint violations under a product measure may not be efficiently expressible and computable in closed form. They may depend on polynomially many terms (for a polynomial of a sufficiently large degree) or may even require exponentially many terms to calculate. This is not only the case for the TSP problem. Other related examples include:

- Spanning tree, acyclicity Zheng et al. (2018), and subgraph connectivity constraints.

- Spectral constraints, i.e., constraints on the eigenvalues of induced subgraphs.
- Sequence and ordering constraints, e.g., learning permutations.

Observe that the challenging constraints tend to involve global properties of the solution. Indeed, for the problems that we have been able to provide clear derivations, a key property is that the constraints (and the objective) are locally-decomposable and/or linear combinations of the decision variables. This generally allows us to leverage the linearity of expectation and union bounds over collections of local events (e.g., clause failure in CNFSAT) to bound the probability of infeasible solutions. However, global constraints such as connectivity tend to involve polynomials over the variables which complicates derivations, and in the hardest cases, might even be computationally intractable. This difficulty is also due to the fact that the product measure over n items is supported on 2^n sets. Therefore, the expectation, unless derivable in close form as we have shown already, might have to be estimated.

It is worth noting that for some constraints we may have access to algorithms and heuristics that allow us to efficiently compute some metric of constraint violation. Unfortunately, in those cases, the algorithm in general is a black box and derivatives can clearly not be calculated for backpropagation. A potential approach there is to use stochastic gradient estimation to differentiate through the black box REINFORCE or other related techniques (Grathwohl et al., 2018). Another issue, is that those algorithms are generally defined for discrete inputs and not continuous distributions so it is not clear how to use them as a proxy for the probability of constraint violation that the probabilistic penalty loss requires. This leaves us with a set of challenges that a general framework for unsupervised combinatorial optimization should be able to address in order for it to be applicable to problems with diverse objectives and constraints.

2.8 Improved bounds: local conditions and local search

We are going to discuss how the probabilistic method can be used in a way that captures problem structure in order to alleviate some of the difficulties discussed in 2.7.1. This can lead to theoretically stronger guarantees for the large class of constraint satisfaction problems.

In order to describe key structural properties of a constraint satisfaction problem instance, we will utilize the concept of a *dependency graph*. A dependency graph $G_\mathrm{I} = (C_I, E_I)$ encodes the relational structure of constraints in the instance I, where each node c_i represents a constraint, and two nodes c_i and c_j are adjacent when $\{\mathrm{vbl}(c_i)\bigcap \mathrm{vbl}(c_j)\} \neq \varnothing$.

Intuitively, for any given formula we can make the following observation: a formula is more highly constrained, and therefore harder to satisfy, when the clauses are densely connected. For example, suppose a variable appears as X_1 in a clause c_1 and as $\neg X_1$ in a clause c_2. In that case, clauses c_1 and c_2 are connected in the dependency graph. Furthermore, fixing X_1 in clause c_1 to True implies that we fix the corresponding literal in C_2 to False. This means now that we have one fewer degree of freedom in C_2 if we want to satisfy it; at least one of the rest of its variables will have to be true.

Building on that intuition, we can use a technique from the probabilistic method toolkit called the *Lovasz Local Lemma* (LLL) Alon and Spencer (2004) which can provide a certificate of satisfiability for constraint satisfaction problems by exploiting the sparsity of the dependency graph. The most well-known formulation of the lemma states that if bad events b_i occur with bounded probability p, i.e., $P(S \in b_i) \leq p$ for $i = 1, 2, \ldots, m$ and if

$$p \leq \frac{1}{e(\Delta+1)}, \tag{2.15}$$

then $P(S \in \bigcap_{i \in C} \bar{b_i}) > 0$, where Δ is the maximum degree of the dependency graph. In other words, if bad events are not very likely and their dependencies are sufficiently sparse, a satisfying assignment of variables exists. Here, the importance of the dependency graph becomes explicit. Compared to the union bound that scales with the number of clauses, the LLL bound scales with the maximum degree of the dependency graph.

Let us consider again the 3-SAT example from before, where each variable is decided based on a fair coin toss. In that case, recall that $P(S \in b_i) = \frac{1}{2^3}$ for any i. From the LLL, we can argue that if

$$\frac{1}{2^3} \leq \frac{1}{e(\Delta+1)} \tag{2.16}$$

then the formula is satisfiable. This also means that we can extract a sufficient condition for satisfiability that depends on the maximum degree. More concretely, from 2.16, we can see that if $\Delta \leq 8/e - 1$, then the instance is satisfiable. For 3-SAT using uniform probabilities over the variables we see that this yields a rather weak condition, as it demands that each clause is connected with at most one other clause. On the other hand, extending this argument for k-SAT for $k > 3$ can lead to much more useful bounds, i.e., for arbitrary k we get $e(\Delta+1) \leq 2^k$. So for uniform assignments, as long as the maximum degree of the dependency graph does not grow faster than 2^k, the LLL could effectively provide satisfiability certificates.

Comparison with the union bound. To highlight the utility of the symmetric LLL compared to the Union bound we may consider the following argument. Since the union bound requires $P(S \in \bigcup_{i=1}^{m} b_i) < 1$, an extrapolation of the bound to a symmetric local condition would be to require $p(S \in b_i) < \frac{1}{m}$. Comparing with the bound of $p(S \in b_i) \leq 1/e(\Delta+1)$ it is easy to see that when $\Delta < \frac{m-e}{e}$ the symmetric LLL yields a strictly easier condition to satisfy for each event b_i.

Improving over the symmetric LLL. Unfortunately, while more promising than the union bound, this bound still has some clear weaknesses, which can be evident from the 3-SAT example. Namely, the maximum degree of the dependency graph may lead to quite strict conditions that are hard to satisfy in practice. In the 3-SAT case, a maximum degree of 1 is unrealistic for any case of practical interest. More generally, when one highly connected node exists in an otherwise sparse dependency graph we obtain a rather strict bound. An additional problem is that this version of the local lemma imposes the same condition for all events b_i. Ideally, we would like to tailor each bound to the local dependency structure of each event. This leads us to the general version of the lemma, known as the *asymmetric* Lovasz Local Lemma which can rectify those shortcomings.

Theorem 4 (Asymmetric Lovász local lemma (Spencer, 1977))**.** Let $\lambda_1, \lambda_2, \ldots, \lambda_m \in [0,1)$ be parameters associated with bad events $b_1, b_2, \ldots, b_m$ and N_i denote the neighborhood of b_i in the dependency graph. If

$$P(S \in b_i) \leq \lambda_i \prod_{j \in \mathcal{N}_i} (1 - \lambda_j), \tag{2.17}$$

then

$$P(S \in \bigcap_{i=1^m} \bar{b_i}) = \prod_{j=1}^{m} (1 - \lambda_j) > 0. \tag{2.18}$$

At first glance, the condition may appear arbitrary due to the presence of additional free parameters. As long as an assignment of values is found for each λ_i that satisfies each inequality, the probabilistic certificate may be recovered. The second thing to note is that the bound now is tailored to each event, depending on the choice of the parameters. Intuitively, the right-hand side of 2.17 parametrizes an independence condition. Suppose that the dependency graph contains only isolated nodes. Then we may freely pick λ_i arbitrarily close to 1 which will guarantee satisfiability for any formula. However, as soon as dependencies are introduced, if λ_i values get arbitrarily close to 1, the bound of $P(b_j)$ for $j \sim i$ in the dependency graph will get arbitrarily close to 0, making it effectively impossible to satisfy. This decomposition of dependencies into local condition is precisely what makes the asymmetric LLL more versatile. Indeed, the asymmetric version has yielded proofs in settings where the symmetric version fails, like the case of proving the existence of certain constrained (frugal) graph colorings (Hind et al., 1997) where the symmetric LLL is inadequate precisely because of the existence of a few highly connected nodes on an otherwise sparse dependency graph.

On the choice of parameters of the asymmetric LLL. It is important to note that the choice of parameters λ_i for each event b_i can lead to significantly different bounds. This is crucial both in a classical context for providing proofs of existence but also important in the sense that the probability of finding the desired object from the LLL depends directly on the parameters λ. Let us briefly consider a couple of different formulations that follow from 2.17.

Corollary 3. Let d_i be the degree of clause c_i in the dependency graph G_I. If

$$p(b_i) \leq \frac{1}{d_i + 1} \prod_{j \in \mathcal{N}_i} \left(\frac{d_j}{d_j + 1} \right), \tag{2.19}$$

then $P(S \in \bigcap_{i \in \mathcal{C}} \bar{b_i}) = \prod_{j=1}^{m} \left(\frac{1}{d_i+1}\right) > 0$.

See proof in Appendix A.1.

This bound explicitly encodes the sparsity structure of each clause. This bound can also make clear the connection between the asymmetric LLL and its symmetric version. Indeed, we may substitute $d_i = \Delta$ in the above abound. Then

$$\left(\prod_{j=1}^{\Delta} \frac{\Delta}{\Delta+1}\right) \to \frac{1}{e}, \text{ as } \Delta \to \infty.$$

Plugging this in 2.19 condition yields the condition of the symmetric LLL. This shows how the symmetric condition is a weaker version of the asymmetric condition and is obtained by taking an upper bound on the degrees of each clause in the dependency graph.

Next, consider a different form of the local lemma which is of computational interest.

Corollary 4 (Dobrushin's condition (Scott and Sokal, 2005))**.** Let $y_1, y_2, \dots, y_m > 0$ be parameters corresponding to bad events $b_1, b_2, \dots, b_m$. Let $P(b_i)$ be the probability of a bad event b_i and let $\mathcal{N}_i^+$. If for all b_i, we have

$$P(S \in b_i) \le \frac{y_i}{\prod_{j \in \mathcal{N}^+} (y_j + 1)},$$

Then $P(S \in \bigcap_{i \in \mathcal{C}} \bar{b_i}) > 0$.

For the proof, see appendix A.1.

Observe that the parameters of each bound are not constrained in the $[0,1)$ interval. This may be preferable from a computational standpoint, as the products in the original asymmetric LLL can lead to numerical instability and machine precision issues when the parameters λ_i are numbers close to 0.

Constructive and algorithmic aspects of the local lemma. While proving that the constraints can be satisfied may be enough in certain applications like model checking, it is often useful to find the variable assignment that satisfies them. In their seminal work, Moser and Tardos (2010) showed that if the conditions of the LLL are satisfied, then the variable assignment can be efficiently found with a simple local resampling routine. The Moser-Tardos algorithm proceeds as follows:

1. Sample a variable assignment according to the learned distribution $\mathcal{D}$.
2. Select arbitrarily a constraint c_i violated by the assignment and resample $\text{vbl}(c_i)$ according to $\mathcal{D}$.

3. Repeat step 2 until no violated constraints remain.

The constructive LLL guarantees that this algorithm will terminate whenever the conditions of the lemma are satisfied. Let T be the total number of resampling steps the algorithm takes until all constraints are repaired. Moser and Tardos showed that

$$\mathbb{E}[T] \leq \sum_{b_i \in \mathcal{B}} \frac{\lambda_i}{1-\lambda_i}, \tag{2.20}$$

which provides a running time certificate.

Learning with local bounds. The probabilistic and algorithmic aspects of the LLL lend themselves to a versatile toolkit for learning algorithm design. Here we briefly describe how one could leverage the bound to efficiently solve instances while obtaining performance guarantees for constraint satisfaction problems. We maintain the standard setup where a set of variables S is assigned the value True and the rest of the variables are set to False. Then the aim is to find S such that all constraints are satisfied. As before, our goal is to learn a parametric distribution $\mathcal{D} = g_\theta(I)$ for each problem instance I which contains such a set with some positive probability.

There are many possible approaches to learning with local bounds. For example, given a dependency graph G, we may train a neural network with

$$\ell(\mathcal{D}; G) := \sum_{i=1}^{m} \sigma\left(P(b_i) - \lambda_i \prod_{j \in N_i} (1-\lambda_j)\right). \tag{2.21}$$

Here, σ is a non-linearity (e.g., relu). By minimizing the loss the network the network finds a distribution that satisfies the conditions of the LLL and proves that there exists a variable assignment that satisfies all the constraints of the instance. Since λ_i are free parameters in the loss function, their choice will be crucial for the success of the neural network. They may be manually chosen based on properties of the input as in 2.19. Alternatively, the free parameters could also be jointly learned with the distribution over the variables with another neural network, i.e., one could opt for learnable y_i from corollary 4. Here it is important to note that in this approach, the neural network may generate a certificate of satisfiability but to recover the satisfying assignment one would then have to use the learned distribution on the Moser-Tardos local search algorithm. In that sense, this approach could be viewed as parametrizing a local search heuristic, similar to the work by Yolcu and Poczos (2019). Such techniques have been successful in the past and the added benefit here would be that one could obtain explicit guarantees that the solution will be found as well as guarantees on the running time of the heuristic via the inequality 2.20. Furthermore, the Moser-Tardos algorithm may be efficiently parallelized, leading to further performance improvements in practice.

More powerful local conditions and additional benefits. It has been noted that the Moser-Tardos algorithm tends to "outperform" the asymmetric LLL in practice. That is, in instances where the asymmetric LLL bound may fail, the Moser-Tardos algorithm may still successfully recover a solution (Catarata et al., 2017). Indeed, the asymmetric LLL is not the most powerful local condition in this context. Other examples include the cluster expansion lemma (Bissacot et al., 2011) and Shearer's criterion (Shearer, 1985). Shearer's criterion is the best known bound and relies on the (generally intractable) function $q_T(G,\mathcal{D})$ which is given by

$$q_T(G,\mathcal{D}) = \sum_{J\in \text{Indep}(G), T\subseteq J} (-1)^{|J|-|T|} \prod_{i\in J} p_i,$$

where $\text{Indep}(G)$ is the set of all independent sets of the graph G. Then Shearer's criterion for a set of m bad events is the following:

Theorem 5 (Shearer's criterion)**.** Let G be a dependency graph on m nodes and $P(S \in b_i)$ for $i = 1, 2, \ldots, m$ the non-zero probabilities of bad events according to a variable distribution $\mathcal{D}$. The following are equivalent:

- $P(S \in \bigcap_{i=1}^m \overline{b_i})$ >0.
- $q_T(G,\mathcal{D}) > 0$ for all $T \in \text{Indep}(G)$.

Observe that Shearer's criterion leverages the independence structure of the graph. Note that the property in the second bullet point cannot be efficiently checked as it requires brute force enumeration of the independent sets of the graph which requires exponential time. On the other hand, it turns out that the Moser-Tardos algorithm will match the effectiveness of the Shearer condition, i.e., the following holds

Theorem 6 (Kolipaka and Szegedy (2011))**.** Let T be the total number of resampling steps that the Moser-Tardos algorithm takes when resampling on a dependency graph G according to a variable distribution $\mathcal{D}$. If Shearer's criterion holds for $\mathcal{D}$, then we have the following:

$$\mathbb{E}[T] \leq \sum_{i=1}^{m} \frac{q_{\{i\}}(G,\mathcal{D})}{q_{\varnothing}(G,\mathcal{D})}.$$

This guarantees that the Moser-Tardos algorithm is effective up to Shearer's criterion. It also provides an explanation regarding the empirical discrepancy between the effectiveness of the asymmetric LLL and the ability of the Moser-Tardos algorithm to solve instances.

Other concentration inequalities. Another possibility towards improving the probabilistic penalty loss is to leverage concentration inequalities in order to further guide the learning process towards high quality solutions. For instance, consider the simple example of Chebyshev's inequality with an unconstrained objective function $f: 2^n \rightarrow \mathbb{R}$ with variance σ^2

$$P(|f(S) - \mathbb{E}[f(S)]| \geq \alpha\sigma) \leq \frac{\mathbb{E}[(f(S) - \mathbb{E}[f(S))^2]}{(\alpha\sigma)^2}.$$

Inevitably, were such a component to be introduced in the loss function for training a neural network, we would need to calculate higher-order moment terms like $\mathbb{E}[f(S)^2]$. Unfortunately, those calculations become increasingly challenging under a product measure, and compact closed-form expressions are not generally available for arbitrary objectives. Finding a coherent and efficient approach to higher-order terms that can yield differentiable losses for a sufficiently wide class of problems is a promising avenue for future research.

2.9 Conclusion

We have presented a principled framework for solving constrained combinatorial problems on graphs that utilizes a probabilistic argument to provide solution quality guarantees. We have demonstrated how this approach can be applied to a plethora of combinatorial problems. We have also discussed in detail the central limitations of the framework and more sophisticated tools of the probabilistic method that could lead to significant improvements. In the next chapter, we will see how we may overcome some of the outstanding obstacles that we have not addressed here and that involve complex constraints and black-box differentiation.

3 Neural set function extensions: Learning with discrete functions in high dimensions

3.1 Introduction

While neural networks are highly effective at solving tasks grounded in basic perception (Chen et al., 2020a; Vaswani et al., 2017), discrete algorithmic and combinatorial tasks such as partitioning graphs, and finding optimal routes or shortest paths have proven more challenging. This is, in part, due to the difficulty of integrating discrete operations into neural network architectures (Battaglia et al., 2018; Bengio et al., 2021; Cappart et al., 2021a). One immediate difficulty with functions on discrete spaces is that they are not amenable to standard gradient-based training. Another is that discrete functions are typically expressed in terms of scalar (e.g., Boolean) variables for each item (e.g., node, edge to be selected), in contrast to the high-dimensional and continuous nature of neural networks' internal representations. A natural approach to addressing these challenges is to carefully choose a function on a continuous domain that *extends* the discrete function, and can be used as a drop-in replacement.

There are several important desiderata that such an extension should satisfy in order to be suited to neural network training. First, an extension should be valid, i.e., agree with the discrete function on discrete points. It should also be amenable to gradient-based optimization, and should avoid introducing spurious minima. Beyond these requirements, there is one additional critical consideration. In both machine learning and optimization, it has been observed that high-dimensional representations can make problems "easier". For instance, neural networks rely on high-dimensional internal representations for representational power and to allow information to flow through gradients, and performance suffers considerably when undesirable low-dimensional bottlenecks are introduced into network architectures (Belkin et al., 2019; Veličković and Blundell, 2021). In optimization, *lifting* to higher-dimensional spaces can make the problem more well-behaved (Goemans and Williamson, 1995; Shawe-Taylor et al., 2004; Du et al., 2018). Therefore, extending discrete functions to *high-dimensional* domains may be critical to the effectiveness of the resulting learning process, yet remains largely an open problem.

With those considerations in mind, we propose a framework for constructing extensions of discrete set functions onto high-dimensional continuous spaces. The core idea is to view a continuous point $\mathbf{x}$ in space as an expectation over a distribution (that depends on $\mathbf{x}$) supported on a few carefully chosen discrete points, to retain tractability. To evaluate the discrete function at $\mathbf{x}$, we compute the expected value of the set function over this distribution. The method resulting from a principled formalization of this idea is computationally efficient and addresses the key challenges of building continuous extensions. Namely, our extensions allow gradient-based optimization and address the dimensionality concerns, allowing any function on sets to be used as a computation step in a neural network.

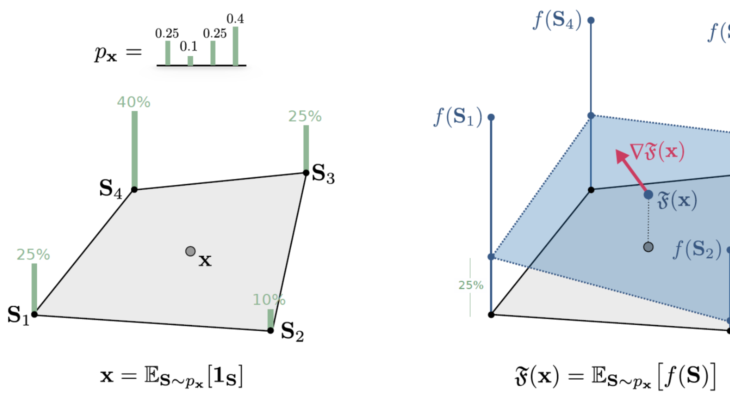
First, to enable gradient computations, we present a method based on a linear programming (LP) relaxation for constructing extensions on continuous domains where exact gradients can be computed using standard automatic differentiation software (Abadi et al., 2016; Bastien et al., 2012; Paszke et al., 2019). Our approach allows task-specific considerations (e.g., a cardinalilty constraint) to be built into the extension design. While our initial LP formulation handles gradients, and is a natural formulation for explicitly building extensions, it replaces discrete Booleans with scalars in the unit interval $[0,1]$, and hence does not yet address potential dimensionality bottlenecks. Second, to enable higher-dimensional representations, we take inspiration from classical SDP relaxations, such as the celebrated Goemans-Williamson maximum cut algorithm (Goemans and Williamson, 1995), which recast low-dimensional problems in high-dimensions. Specifically, our key contribution is to develop an SDP analog of our original LP formulation, and show how to *lift* LP-based extensions into a corresponding high-dimensional SDP-based extensions. Our general procedure for lifting low-dimensional representations into higher dimensions aligns with the neural algorithmic reasoning blueprint (Veličković and Blundell, 2021), and suggests that classical techniques such as SDPs may be effective tools for combining deep learning with algorithmic processes more generally.

3.2 Problem Setup

Consider a ground set $[n] = \{1, \dots, n\}$ and an arbitrary function $f : 2^{[n]} \to \mathbb{R} \cup \{\infty\}$ defined on subsets of $[n]$. For instance, f could determine if a set of nodes or edges in a graph has some structural property, such as being a path, tree, clique, or independent set (Bello et al., 2016; Cappart et al., 2021a). Our aim is to build neural networks that use such discrete functions f as an intermediate layer or loss. In order to produce a model that is trainable using standard auto-differentiation software, we consider a continuous domain $\mathcal{X}$ onto which we would like to extend f, with sets embedded into $\mathcal{X}$ via an injective map $e : 2^{[n]} \to \mathcal{X}$. For instance, when $\mathcal{X} = [0,1]^n$ we may take $e(S) = \mathbf{1}_S$, the Boolean vector whose ith entry is 1 if $i \in S$, and 0 otherwise. Our approach is to design an extension

$$\mathfrak{F} : \mathcal{X} \to \mathbb{R}$$

of f and consider the neural network $\mathrm{NN}_2 \circ \mathfrak{F} \circ \mathrm{NN}_1$ (if f is used as a loss, NN_2 is simply the identity). To ensure that the extension is *valid* and amenable to automatic differentiation, we require that 1) it agrees with f on all discrete points: $\mathfrak{F}(e(S)) = f(S)$ for all $S \subseteq [n]$ with $f(S) < \infty$, and 2) $\mathfrak{F}$ is continuous.



$p_{\mathbf{x}} =$
0.25
0.1
0.25
0.4
40%
25%
$\mathbf{S}_4$
$\mathbf{S}_3$
25%
$\mathbf{x}$
$\mathbf{S}_1$
10%
$\mathbf{S}_2$
$\mathbf{x} = \mathbb{E}_{\mathbf{S}\sim p_{\mathbf{x}}}[\mathbf{1_S}]$
$f(\mathbf{S}_4)$
$f(\mathbf{S}_3)$
$f(\mathbf{S}_1)$
$\nabla\mathfrak{F}(\mathbf{x})$
$\mathfrak{F}(\mathbf{x})$
$f(\mathbf{S}_2)$
25%
$\mathfrak{F}(\mathbf{x}) = \mathbb{E}_{\mathbf{S}\sim p_{\mathbf{x}}}[f(\mathbf{S})]$

Figure 3.1: **SFEs:** Fractional points $\mathbf{x}$ are reinterpreted as expectations $\mathbf{x} = \mathbb{E}_{S\sim p_{\mathbf{x}}}[\mathbf{1}_S]$ over the distribution $p_{\mathbf{x}}(S)$ on sets. A value is assigned at $\mathbf{x}$ by exchanging the order of f and the expectation: $\mathfrak{F}(\mathbf{x})_{S\sim p_{\mathbf{x}}}[f(S)]$. Unlike f, the extension $\mathfrak{F}$ is amenable to gradient-based optimization.

There is a rich existing literature on extensions of functions on discrete domains, particularly in the context of discrete optimization (Lovász, 1983; Grötschel et al., 1981; Calinescu et al., 2011; Vondrák, 2008; Bach, 2019; Obozinski and Bach, 2012; Tawarmalani and Sahinidis, 2002). These works provide promising tools to reach our goal of neural network training. Building on these, our method is the first to use semi-definite programming (SDP) to combine neural networks with set functions. There are, however, different considerations in the neural network setting as compared to optimization. The optimization literature often focuses on a class of set functions and aims to build extensions with desirable optimization properties, particularly convexity. We do not focus on convexity, aiming instead to develop a formalism that is as flexible as possible. Doing so maximizes the applicability of our method, and allows extensions adapted to task-specific desiderata (see Section 3.3.1).

3.3 Scalar Set Function Extensions

We start by presenting a general framework for extending set functions onto $\mathcal{X} = [0,1]^n$, where a set $S \subseteq [n]$ is viewed as the Boolean indicator vector $e(S) = \mathbf{1}_S \in \{0,1\}^n$ whose ith entry is 1 if $i \in S$ and 0 otherwise. We call extensions onto $[0,1]^n$ *scalar* since each item i is represented by a single scalar value—the ith coordinate of $\mathbf{x} \in \mathcal{X}$. These scalar extensions will become the core building blocks in developing high-dimensional extensions in Section 3.4.

A classical approach to extending discrete functions on sets represented as Boolean indicator vectors $\mathbf{1}_S$ is by computing the convex-envelope, i.e., the point-wise supremum over linear functions that lower bound f (Falk and Hoffman, 1976; Bach, 2019). Doing so yields a convex function whose value at a point $\mathbf{x} \in [0,1]^n$ is the solution of the following linear program (LP):

$$\widetilde{\mathfrak{F}}(\mathbf{x}) = \max_{\mathbf{z}, b \in \mathbb{R}^n \times \mathbb{R}} \{\mathbf{x}^\top \mathbf{z} + b\} \text{ subject to } \mathbf{1}_S^\top \mathbf{z} + b \leq f(S) \text{ for all } S \subseteq [n]. \qquad \text{(primal LP)}$$

The set $\mathcal{P}_f$ of all feasible solutions $(\mathbf{z}, b)$ is known as the *(canonical) polyhedron of f* (Obozinski and Bach, 2012) and can be seen to be non-empty by taking the coordinates of $\mathbf{z}$ to be sufficiently small (possibly negative). Variants of this optimization program are frequently encountered in the theory of matroids and submodular functions (Edmonds, 2003) where $\mathcal{P}_f$ is commonly known as the *submodular polyhedron* (see Appendix 3.7 for an extended discussion). By strong duality, we may solve the primal LP by instead solving its dual:

$$\widetilde{\mathfrak{F}}(\mathbf{x}) = \min_{\{y_S \geq 0\}_{S \subseteq [n]}} \sum_{S \subseteq [n]} y_S f(S) \text{ subject to } \sum_{S \subseteq [n]} y_S \mathbf{1}_S = \mathbf{x}, \ \sum_{S \subseteq [n]} y_S = 1, \ \text{ for all } S \subseteq [n], \qquad \text{(dual LP)}$$

whose optimal value is the same as the primal LP. The dual LP is always feasible (see e.g., the Lovász extension in Section 3.3.1). However, $\widetilde{\mathfrak{F}}$ does not necessarily agree with f on discrete points in general, unless the function is convex-extensible (Murota, 1998).

To address this important missing piece, we relax our goal from solving the dual LP to instead seeking a *feasible* solution to the dual LP that *is* an extension of f. Since the dual LP is defined for a fixed $\mathbf{x}$, a feasible solution must be a function $y_S = p_{\mathbf{x}}(S)$ of $\mathbf{x}$. If $p_{\mathbf{x}}$ were to be continuous and a.e. differentiable in $\mathbf{x}$ then the value $\sum_S p_{\mathbf{x}}(S) f(S)$ attained by the dual LP would also be continuous and a.e. differentiable in $\mathbf{x}$ since gradients flow through the coefficients $y_S = p_{\mathbf{x}}(S)$, while $f(S)$ is treated as a constant in $\mathbf{x}$. This leads us to the following definition:

Definition (Scalar SFE)**.** A scalar SFE $\mathfrak{F}$ of f is defined at a point $\mathbf{x} \in [0,1]^n$ by coefficients $p_{\mathbf{x}}(S)$ such that $y_S = p_{\mathbf{x}}(S)$ is a feasible solution to the dual LP. The extension value is given by

$$\mathfrak{F}(\mathbf{x}) \triangleq \sum_{S \subseteq [n]} p_{\mathbf{x}}(S) f(S)$$

and we require the following properties to hold for all $S \subseteq [n]$: 1) $p_{\mathbf{x}}(S)$ is a continuous function of $\mathbf{x}$ and 2) $\mathfrak{F}(\mathbf{1}_S) = f(S)$ for all $S \subseteq [n]$.

Efficient evaluation of $\mathfrak{F}$ requires that $p_{\mathbf{x}}(S)$ is supported on a small collection of carefully chosen sets S. This choice is a key inductive bias of the extension, and Section 3.3.1 gives many examples with only $O(n)$ non-zero coefficients. Examples include well-known extensions, such as the Lovász extension, as well as a number of novel extensions, illustrating the versatility of the SFE framework.

Thanks to the constraint $\sum_S y_S = 1$ in the dual LP, scalar SFEs have a natural probabilistic interpretation. An SFE is defined by a probability distribution $p_{\mathbf{x}}$ such that fractional points $\mathbf{x}$ can be written as an expectation $\mathbb{E}_{S \sim p_{\mathbf{x}}}[\mathbf{1}_S] = \mathbf{x}$ over discrete points using $p_{\mathbf{x}}$. The extension itself can be viewed as arising from exchanging f and the expectation operation: $\mathfrak{F}(\mathbf{x}) = \mathbb{E}_{S \sim p_{\mathbf{x}}}[f(S)]$. This interpretation is summarized in Figure 3.1.

Scalar SFEs also enjoy the property of not introducing any spurious minima. That is, the minima of $\mathfrak{F}$ coincide with the minima of f up to convex combinations. This property is especially important when training models of the form $f \circ \mathrm{NN}_1$ (i.e., f is a loss function) since $\mathfrak{F}$ will guide the network NN_1 towards the same solutions as f.

Proposition 1 (Scalar SFEs have no bad minima)**.** If $\mathfrak{F}$ is a scalar SFE of f then:

1. $\min_{\mathbf{x}\in\mathcal{X}} \mathfrak{F}(\mathbf{x}) = \min_{S\subseteq[n]} f(S)$
2. $\operatorname{argmin}_{\mathbf{x}\in\mathcal{X}} \mathfrak{F}(\mathbf{x}) \subseteq \mathrm{Hull}\big(\operatorname{argmin}_{\mathbf{1}_S : S\subseteq[n]} f(S)\big)$

See Appendix 3.8 for proofs.

Obtaining set solutions. Given an architecture $\mathfrak{F}\circ\mathrm{NN}_1$ and input problem instance G, we often wish to produce sets as outputs at inference time. To do this, we simply compute $\mathbf{x} = \mathrm{NN}_1(G)$, and select the set S in $\mathrm{supp}_S\{p_{\mathbf{x}}(S)\}$ with the smallest value $f(S)$. This can be done efficiently if, as is typically the case, the cardinality of $\mathrm{supp}_S\{p_{\mathbf{x}}(S)\}$ is small.

3.3.1 Constructing Scalar Set Function Extensions

A key characteristic of scalar SFEs is that there are many potential extensions of any given f. In this section, we provide examples of scalar SFEs, illustrating the capacity of the SFE framework for building knowledge about f into the extension. See Appendix **??** for all proofs and further discussion.

Lovász extension. Re-indexing the coordinates of $\mathbf{x}$ so that $x_1 \geq x_2 \ldots \geq x_n$, we define $p_{\mathbf{x}}$ to be supported on the sets $S_1 \subseteq S_2 \subseteq \cdots \subseteq S_n$ with $S_i = \{1, 2, \ldots, i\}$ for $i = 1, 2, \ldots, n$. The coefficient are defined as $y_{S_i} = p_{\mathbf{x}}(S_i) := x_i - x_{i+1}$ and $p_{\mathbf{x}}(S) = 0$ for all other sets. The resulting *Lovász extension*—known as the *Choquet integral* in decision theory (Choquet, 1954; Marichal, 2000)—is a key tool in combinatorial optimization due to a seminal result: the Lovász extension is convex if and only if f is submodular (Lovász, 1983), implying that submodular minimization can be solved in polynomial-time (Grötschel et al., 1981).

Bounded cardinality Lovász extension. A collection $\{S_i\}_{i=1}^n$ of subsets of $[n]$ can be encoded in an $n \times n$ matrix $\mathbf{S} \in \{0,1\}^{n\times n}$ whose ith column is $\mathbf{1}_{S_i}$. In this notation, the dual LP constraint $\sum_{S\subseteq[n]} y_S \mathbf{1}_S = \mathbf{x}$ can be written as $\mathbf{Sp} = \mathbf{x}$, where the ith coordinate of $\mathbf{p}$ defines $p_{\mathbf{x}}(S_i)$. The *bounded cardinality* extension generalizes the Lovász extension to focus only on sets of cardinality at most $k \leq n$. Again, re-index $\mathbf{x}$ so that $x_1 \geq x_2 \ldots \geq x_n$. Use the first k sets $S_1 \subseteq S_2 \subseteq \cdots \subseteq S_k$, where $S_i = \{1, 2, \ldots, i\}$, to populate the first k columns of matrix $\mathbf{S}$. We add further $n-k$ sets: $S_{k+i} = \{j + i \mid j \in S_k\}$ for $i = 1, \ldots, n-k$, to fill the rest of $\mathbf{S}$. Finally, $p_{\mathbf{x}}(S_i)$ can be analytically calculated from $\mathbf{p} = \mathbf{S}^{-1}\mathbf{x}$, where $\mathbf{S}$ is invertible since it is a Toeplitz banded upper triangular matrix.

Permutations and involutory extensions. We use the same $\mathbf{S}, \mathbf{p}$ notation. Let $\mathbf{S}$ be an elementary permutation matrix. Then it is involutory, i.e., $\mathbf{SS} = \mathbf{I}$, and we may easily determine $\mathbf{p} = \mathbf{Sx}$

given $\mathbf{S}$ and $\mathbf{x}$. Note that $p_{\mathbf{x}}(S_i) = \mathbf{p}_i$ must be non-negative since $\mathbf{x}$ and $\mathbf{S}$ are non-negative entry-wise. Finally, restricting $\mathbf{x}$ to the n-dimensional Simplex guarantees that $\|\mathbf{p}\|_1 \leq 1$, which ensures $p_{\mathbf{x}}$ is a probability distribution (any remaining mass is placed on the empty set). The extension property can be guaranteed on singleton sets as long as the chosen permutation admits a fixed point at the argmax of $\mathbf{x}$. Any elementary permutation matrix $\mathbf{S}$ with such a fixed point yields a valid SFE.

Singleton extension. Consider a set function f for which $f(S) = \infty$ unless S has cardinality one. To ensure $\mathfrak{F}$ is finite valued, $p_{\mathbf{x}}$ must be supported only on the sets $S_i = \{i\}$, $i = 1, \ldots, n$. Assuming $\mathbf{x}$ is sorted so that $x_1 \geq x_2 \ldots \geq x_n$, define $p_{\mathbf{x}}(S_i) = x_i - x_{i+1}$. It is shown in Appendix **??** that this defines a scalar SFE, except for the dual LP feasibility. However, when using $\mathfrak{F}$ as a loss function, minimization drives $\mathbf{x}$ towards the minima $\min_{\mathbf{x}} \mathfrak{F}(\mathbf{x})$ which *are* dual feasible. So dual infeasibility is benign in this instance and we approach the feasible set from the outside.

Multilinear extension. The multilinear extension, widely used in combinatorial optimization (Calinescu et al., 2011), is supported on all sets with coefficients $p_{\mathbf{x}}(S) = \prod_{i \in S} x_i \prod_{i \notin S}(1 - x_i)$, the product distribution. In general, evaluating the multilinear extension exactly requires 2^n calls to f, but for several interesting set functions, e.g., graph cut, set cover, and facility location, it can be computed efficiently in $\widetilde{\mathcal{O}}(n^2)$ time (Iyer et al., 2014).

3.4 Neural Set Function Extensions

This section builds on the scalar SFE framework—where each item i in the ground set $[n]$ is represented by a single scalar—to develop extensions that use high-dimensional embeddings to avoid introducing low-dimensional bottlenecks into neural network architectures. The core motivation that lifting problems into higher dimensions can make them easier is not unique to deep learning. For instance, it also underlies kernel methods (Shawe-Taylor et al., 2004) and the *lift-and-project* method for integer programming (Lovász and Schrijver, 1991).

Our method takes inspiration from prior successes of semi-definite programming for combinatorial optimization (Goemans and Williamson, 1995) by extending onto $\mathcal{X} = \mathbb{S}_+^n$, the set of $n \times n$ positive semi-definite (PSD) matrices. With this domain, each item is represented by a vector, not a scalar.

3.4.1 Lifting Set Function Extensions to Higher Dimensions

We embed sets into $\mathbb{S}_+^n$ via the map $e(S) = \mathbf{1}_S \mathbf{1}_S^\top$. To define extensions on this matrix domain, we translate the linear programming approach of Section 3.3 into an analogous SDP formulation:

$$\max_{\mathbf{Z} \succeq 0, b \in \mathbb{R}} \{\mathrm{Tr}(\mathbf{X}^\top \mathbf{Z}) + b\} \text{ subject to } \frac{1}{2}\mathrm{Tr}((\mathbf{1}_S \mathbf{1}_T^\top + \mathbf{1}_T \mathbf{1}_S^\top)\mathbf{Z}) + b \leq f(S \cap T) \text{ for all } S, T \subseteq [n], \tag{primal SDP}$$

where we switch from lower case letters to upper case since we are now using matrices. Next, we show that this choice of primal SDP is a natural analog of the original LP that provides the right correspondences between vectors and matrices by proving that primal LP feasible solutions correspond to primal SDP feasible solutions with the same objective value (see Appendix 3.7 for a discussion on the SDP and its dual). To state the result, note that the embedding $e(S) = \mathbf{1}_S\mathbf{1}_S^\top$ is a particular case of the correspondence $\mathbf{x} \in [0,1]^n \mapsto \sqrt{\mathbf{x}}\sqrt{\mathbf{x}}^\top$.

Proposition 2. (Containment of LP in SDP) For any $\mathbf{x} \in [0,1]^n$, define $\mathbf{X} = \sqrt{\mathbf{x}}\sqrt{\mathbf{x}}^\top$ with the square-root taken entry-wise. Then, for any $(\mathbf{z}, b) \in \mathbb{R}_+^n \times \mathbb{R}$ that is primal LP feasible, the pair $(\mathbf{Z}, b)$ where $\mathbf{Z} = \mathrm{diag}(\mathbf{z})$, is primal SDP feasible and the objective values agree: $\mathrm{Tr}(\mathbf{X}^\top\mathbf{Z}) = \mathbf{z}^\top\mathbf{x}$.

Proposition 2 establishes that the primal SDP feasible set can be obtained from a linear map of the positive primal LP feasible set, i.e., feasible solutions of the primal LP lead to feasible solutions of the primal SDP. As with scalar SFEs, to define neural SFEs we consider the dual SDP:

$$\min_{\{y_{S,T}\geq 0\}} \sum_{S,T\subseteq[n]} y_{S,T} f(S\cap T) \ \text{ subject to } \ \mathbf{X} \preceq \sum_{S,T\subseteq[n]} \frac{1}{2} y_{S,T}(\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top) \ \text{ and } \sum_{S,T\subseteq[n]} y_{S,T} = 1 \tag{dual SDP}$$

We demonstrate that for suitable $\mathbf{X}$ this SDP has feasible solutions via an explicit construction in Section 3.4.2 [1]. This leads us to define a neural SFE which, as with scalar SFEs, is given by a feasible solution to the dual SDP that satisfies the extension property whose coefficients are continuous in $\mathbf{X}$:

Definition (Neural SFE)**.** A neural set function extension of f at a point $\mathbf{X} \in \mathbb{S}_+^n$ is defined as

$$\mathfrak{F}(\mathbf{X}) \triangleq \sum_{S,T\subseteq[n]} p_{\mathbf{X}}(S,T) f(S\cap T),$$

where $y_{S,T} = p_{\mathbf{X}}(S,T)$ is a feasible solution to the dual SDP and for all $S, T \subseteq [n]$: 1) $p_{\mathbf{X}}(S,T)$ is continuous at $\mathbf{X}$ and 2) it is valid, i.e., $\mathfrak{F}(\mathbf{1}_S\mathbf{1}_S^\top) = f(S)$ for all $S \subseteq [n]$.

3.4.2 Constructing Neural Set Function Extensions

We constructed a number of explicit examples of scalar SFEs in Section 3.3.1. For neural SFEs we employ a different strategy. Instead of providing individual examples of neural SFEs, we develop a single recipe for converting *any* scalar SFE into a corresponding neural SFE. Doing so allows us to build on the variety of scalar SFEs and provides an additional connection between scalar and neural SFEs. In Section 3.5 we show the empirical superiority of neural SFEs over their scalar counterparts.

Our construction is given in the following proposition:

[1] We may also formulate a primal-dual pair that relies entirely on rank one matrices, see 3.7.3 for more details.

Proposition 3. Let $p_{\mathbf{x}}$ induce a scalar SFE of f. For $\mathbf{X} \in \mathbb{S}_+^n$, consider a decomposition $\mathbf{X} = \sum_{i=1}^n \lambda_i \mathbf{x}_i \mathbf{x}_i^\top$ and fix

$$p_{\mathbf{X}}(S,T) = \sum_{i=1}^n \lambda_i \, p_{\mathbf{x}_i}(S) p_{\mathbf{x}_i}(T) \text{ for all } S,T \subseteq [n]. \tag{3.1}$$

Then, $p_{\mathbf{X}}$ defines a neural SFE $\mathfrak{F}$ at $\mathbf{X}$.

See Appendix B.2 for proof. The choice of decomposition will give rise to different extensions. Here, we instantiate our neural extensions using the eigendecomposition of $\mathbf{X}$. Since eigenvectors may not belong to $[0,1]^n$ we reparameterize by first applying a sigmoid function before computing the scalar extension distribution $p_{\mathbf{x}}$. In practice we found that neural SFEs work just as well even without this sigmoid function—i.e., allowing scalar SFEs to be evaluated outside of $[0,1]^n$. The continuity of the neural SFE $\mathfrak{F}$ when using the eigendecomposition follows from a variant of the Davis–Kahan theorem (Yu et al., 2015), which requires the additional assumption that the eigenvalues of $\mathbf{x}$ are distinct. For efficiency, in practice we do not use all n eigenvectors, and use only the k with largest eigenvalue. This is justified by Figure 3.4, which shows that in practical applications $\mathbf{X}$ often has a rapidly decaying spectrum.

Evaluating a neural SFE requires an accessible closed-form expression, the precise form of which depends on the underlying scalar SFE. Further, from the definition of Neural SFEs we see that if a scalar SFE is supported on sets with a property that is closed under intersection (e.g., bounded cardinality), then the supporting sets of the corresponding neural SFE will also inherit that property. This implies that the neural counterparts of the Lovász, bounded cardinality Lovász, and singleton/permutation extensions have the same support as their scalar counterparts. An immediate corollary is that we can easily compute the neural counterpart of the Lovász extension which has a simple closed form:

Corollary 5. For $\mathbf{X} \in \mathbb{S}_+^n$ consider the eigendecomposition $\mathbf{X} = \sum_{i=1}^n \lambda_i \mathbf{x}_i \mathbf{x}_i^\top$. Let $p_{\mathbf{x}_i}$ be as in the Lovász extension: $p_{\mathbf{x}_i}(S_{ij}) = \sigma(x_{i,j}) - \sigma(x_{i,j+1})$, where σ is the sigmoid function, and $\mathbf{x}_i$ is sorted so $x_{i,1} \geq \ldots \geq x_{i,n}$ and $S_{ij} = \{1,\ldots,j\}$, with $p_{\mathbf{x}_i}(S) = 0$ for all other sets. Then, the neural Lovász extension is:

$$\mathfrak{F}(\mathbf{X}) = \sum_{i,j=1}^n \lambda_i p_{\mathbf{x}_i}(S_{ij}) \cdot \left(p_{\mathbf{x}_i}(S_{ij}) + 2 \sum_{\ell:\ell>j} p_{\mathbf{x}_i}(S_{i\ell}) \right) \cdot f(S_{ij}). \tag{3.2}$$

Complexity and obtaining sets as solutions. In general, the neural SFE relies on all pairwise intersections $S \cap T$ of the scalar SFE sets, requiring $O(m^2)$ evaluations of f when the scalar SFE is supported on m sets. However, when the scalar SFE is supported on a family of sets that is closed under intersection—e.g., the Lovász and singleton extensions—the corresponding neural SFE requires only $O(m)$ function evaluations. Discrete solutions can be obtained efficiently by returning the best set out of all scalar SFEs $p_{\mathbf{x}_i}$.

	Maximum Clique				
	ENZYMES	PROTEINS	IMDB-Binary	MUTAG	COLLAB
Straight-through (Bengio et al., 2013)	$0.725_{\pm 0.268}$	$0.722_{\pm 0.26}$	$0.917_{\pm 0.253}$	$0.965_{\pm 0.162}$	$0.856_{\pm 0.221}$
Erdős (Karalias and Loukas, 2020)	$0.883_{\pm 0.156}$	$0.905_{\pm 0.133}$	$0.936_{\pm 0.175}$	$1.000_{\pm 0.000}$	$0.852_{\pm 0.212}$
REINFORCE (Williams, 1992)	$0.751_{\pm 0.301}$	$0.725_{\pm 0.285}$	$0.881_{\pm 0.240}$	$1.000_{\pm 0.000}$	$0.781_{\pm 0.316}$
Lovász scalar SFE	$0.723_{\pm 0.272}$	$0.778_{\pm 0.270}$	$0.975_{\pm 0.125}$	$0.977_{\pm 0.125}$	$0.855_{\pm 0.225}$
Lovász neural SFE	$0.933_{\pm 0.148}$	$0.926_{\pm 0.165}$	$0.961_{\pm 0.143}$	$1.000_{\pm 0.000}$	$0.864_{\pm 0.205}$
	Maximum Independent Set				
	ENZYMES	PROTEINS	IMDB-Binary	MUTAG	COLLAB
Straight-through (Bengio et al., 2013)	$0.505_{\pm 0.244}$	$0.430_{\pm 0.252}$	$0.701_{\pm 0.252}$	$0.721_{\pm 0.257}$	$0.331_{\pm 0.260}$
Erdős (Karalias and Loukas, 2020)	$0.821_{\pm 0.124}$	$0.903_{\pm 0.114}$	$0.515_{\pm 0.310}$	$0.939_{\pm 0.069}$	$0.886_{\pm 0.198}$
REINFORCE (Williams, 1992)	$0.617_{\pm 0.214}$	$0.579_{\pm 0.340}$	$0.899_{\pm 0.275}$	$0.744_{\pm 0.121}$	$0.053_{\pm 0.164}$
Lovász scalar SFE	$0.311_{\pm 0.289}$	$0.462_{\pm 0.260}$	$0.716_{\pm 0.269}$	$0.737_{\pm 0.154}$	$0.302_{\pm 0.238}$
Lovász neural SFE	$0.775_{\pm 0.155}$	$0.729_{\pm 0.205}$	$0.679_{\pm 0.287}$	$0.854_{\pm 0.132}$	$0.392_{\pm 0.253}$

Table 3.1: **Unsupervised neural combinatorial optimization**: Approximation ratios for combinatorial problems. Values closer to 1 are better (↑). Neural SFEs are competitive with other methods, and consistently improve over vector SFEs.

3.5 Experiments

We experiment with SFEs as loss functions in neural network pipelines on discrete objectives arising in combinatorial and vision tasks. For combinatorial optimization, SFEs network training with a continuous version of the objective without supervision. For supervised image classification, they allow us to directly relax the training error instead of optimizing a proxy like cross entropy.

3.5.1 Unsupervised Neural Combinatorial Optimization

We begin by evaluating the suitability of neural SFEs for unsupervised learning of neural solvers for combinatorial optimization problems on graphs. We use the ENZYMES, PROTEINS, IMDB, MUTAG, and COLLAB datasets from the TUDatasets benchmark (Morris et al., 2020), using a 60/30/10 split for train/test/val. We test on two problems: finding maximum cliques, and maximum independent sets. We compare with three neural network based methods. We compare to two common approaches for backpropogating through discrete functions: the REINFORCE algorithm (Williams, 1992), and the Straight-Through estimator (Bengio et al., 2013). The third is the recently proposed probabilistic penalty relaxation (Karalias and Loukas, 2020) for combinatorial optimization objectives. All methods use the same GNN backbone, comprising a single GAT layer (Veličković et al., 2018) followed by multiple gated graph convolution layers Li et al. (2015).

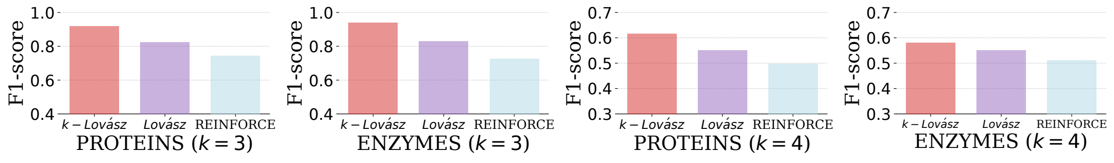


Figure 3.2: **k-clique constraint satisfaction:** higher F1-score is better. The k-bounded cardinality Lovasz extension is better aligned with the task and significantly improves over the Lovász extension.

In all cases, given an input graph $G = (V, E)$ with $|V| = n$ nodes, a GNN produces an embedding for each node: $\mathbf{X} \in \mathbb{R}^{n\times d}$. For scalar SFEs $d = 1$, while for neural SFEs we consider $\mathbf{X}\mathbf{X}^\top$ in order to produce an $n \times n$ PSD matrix, which is passed as input to the SFE $\mathfrak{F}$. The set function f used is problem dependent, which we discuss below. Finally, see Appendix B.5 for training and hyper-parameter optimization details, and Appendix B.4 for details on data, hardware, and software.

Maximum Clique. A set $S \subseteq V$ is a clique of $G = (V, E)$ if $(i, j) \in E$ for all $i, j \in S$. The MaxClique problem is to find the largest set S that is a clique: i.e., $f(S) = |S| \cdot \mathbf{1}\{S \text{ a clique}\}$.

Maximum Independent Set (MIS). A set $S \subseteq V$ is an independent set of $G = (V, E)$ if $(i, j) \notin E$ for all $i, j \in S$. The goal is to find the largest S in the graph that is independent, i.e., $f(S) = |S| \cdot \mathbf{1}\{S \text{ an ind. set}\}$. MIS differs significantly from MaxClique due to its high heterophily.

Results. Table 3.1 displays the mean and standard deviation of the approximation ratio $f(S)/f(S^*)$ of the solver solution S and an optimal S^* on the test set graphs. The neural Lovaśz extension outperforms its scalar counterpart in 8 out of 10 cases, often by significant margins, for instance improving a score of 0.778 on PROTEINS MaxClique to 0.926. The neural SFE proved effective at boosting poor scalar SFE performance, e.g., 0.311 on ENZYMES MIS, to the competitive performance of 0.775. Neural Lovaśz outperformed or equalled and straight-through in 9 out of 10 cases, and the method of Karalias and Loukas (2020) in 6 out of 10.

3.5.2 Constraint Satisfaction Problems

Constraint satisfaction problems ask if there exists a set satisfying a given set of conditions (Kumar, 1992; Cappart et al., 2021b). In this section, we apply SFEs to the k-clique problem: given a graph, determine if it contains a clique of size k or more. We test on the ENZYMES and PROTEINS datasets. Since satisfiability is a binary classification problem we evaluate using F1 score.

Results. Figure 3.2 shows that by specifically searching over sets of size k using the cardinality constrained Lovász extension from Section 3.3.1, we significantly improve performance compared to the Lovász extension, and REINFORCE. This illustrates the value of SFEs in allowing

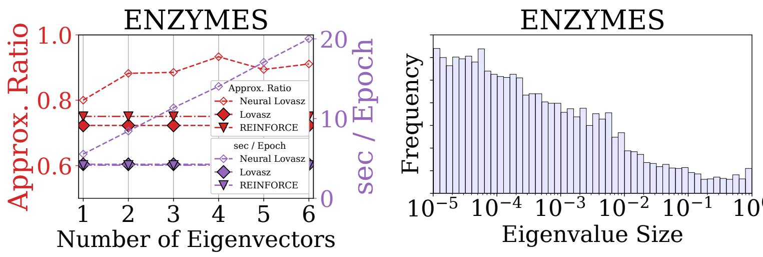


Figure 3.3: **Left:** Runtime and performance of neural SFEs on MaxClique using different numbers of eigenvectors. **Right:** Histogram of spectrum of matrix **X**, outputted by a GNN trained on MaxClique.

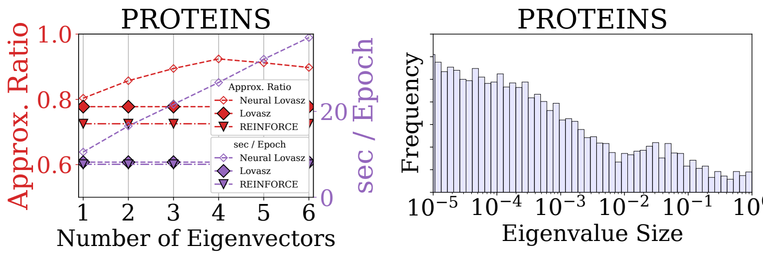


Figure 3.4: **Left:** Runtime and performance of neural SFEs on MaxClique using different numbers of eigenvectors. **Right:** Histogram of spectrum of matrix **X**, outputted by a GNN trained on MaxClique.

task-dependent considerations (in this case a cardinality constraint) to be built into extension design.

3.5.3 Training Error as a Classification Objective

During training the performance of a classifier h is typically assessed using the training error $\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{y_i \neq h(x_i)\}$. Since the training error itself is non-differentiable, it is standard to train h to optimize a differentiable surrogate such as the cross-entropy loss. Here we offer an alternative training method by continuously extending the non-differentiable mapping $\hat{y} \mapsto \mathbf{1}\{y_i \neq \hat{y}\}$. This map is a set function defined on single item sets, so we use the singleton extension (definition in Section 3.3.1). Our goal is to demonstrate that the resulting differentiable loss function closely tracks the training error, and can be used to minimize it. We do not focus on test time generalization. Figure B.1 shows the results. The singleton extension loss (left plot) closely tracks the true training error at the same numerical scale, unlike other common loss functions (see Appendix B.6 for setup details). While we leave further consideration to future work, training error extensions may be useful for model calibration (Kennedy and O'Hagan, 2001) and uncertainty estimation (Abdar et al., 2021).

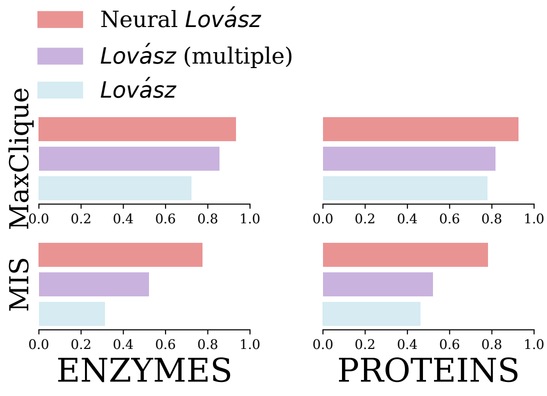


Figure 3.5: Obtaining a neural SFE improves performance over an ensemble of low-dimensional extensions.

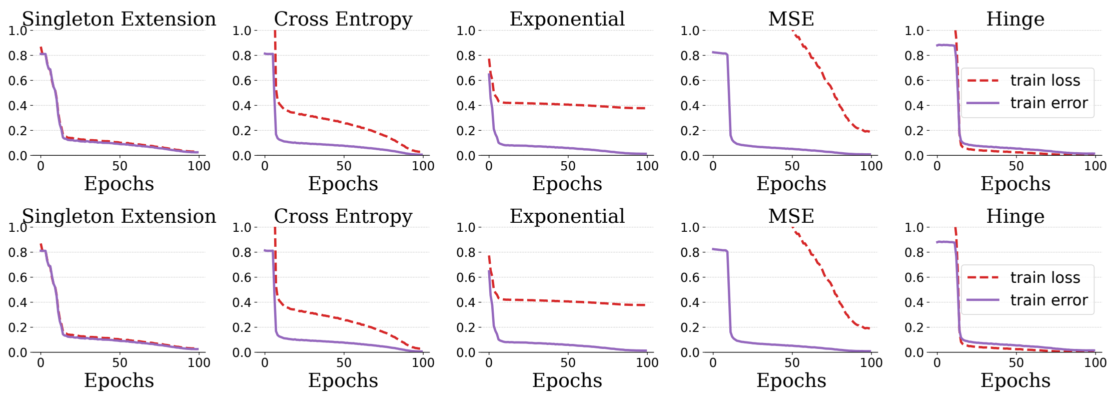


Figure 3.6: Top: CIFAR10. Bottom: SVHN. The singleton extension loss (left) is the only loss that approximates the true non-differentiable training error at the same numerical scale.

3.5.4 Ablations

Number of Eigenvectors. Figure 3.4 compares the runtime and performance of neural SFEs using only the top-k eigenvectors from the eigendecomposition $\mathbf{X} = \sum_{i=1}^{n} \lambda_i \mathbf{x}_i \mathbf{x}_i^\top$ with $k \in \{1,2,3,4,5,6\}$ on the maximum clique problem. For both ENZYMES and PROTEINS, performance increases with k—easily outperforming scalar SFEs and REINFORCE—until saturation around $k = 4$, while runtime grows linearly with k. Histograms of the eigenvalues produced by trained networks show a rapid decay in the spectrum, suggesting that the smaller eigenvalues have little effect on $\mathfrak{F}$.

Comparison to Naive High-Dimensional Extension. We compare neural SFEs to a naive high-dimensional alternative which, given an $n \times d$ matrix $\mathbf{X}$ simply computes a scalar SFE on each column independently and sums them up. This naive function design is not an extension, and the dependence on the d dimensions is linearly separable, in contrast to the complex non-linear interactions between columns of $\mathbf{X}$ in neural SFEs. Figure 3.5 shows that this naive extension, whilst improving over one-dimensional extensions, performs considerably worse than neural SFEs.

3.6 Related Work

Neural combinatorial optimization Our experimental setup largely follows recent work on unsupervised neural combinatorial optimization (Karalias and Loukas, 2020; Schuetz et al., 2022; Xu et al., 2020a; Toenshoff et al., 2021; Amizadeh et al., 2018), where continuous relaxations of discrete objectives are utilized. In that context, it is important to take into account the key conceptual and methodological differences of our approach. For instance, in the unsupervised *Erdős goes neural* (EGN) framework from Karalias and Loukas (2020), the probabilistic relaxation and the proposed choice of distribution can be viewed as instantiating a multilinear extension. As explained earlier, this extension is costly in the general case (since f must be evaluated 2^n times, and summed) but can be computed efficiently in closed form in certain cases. On the other hand, our extension framework offers multiple options for efficiently computable extensions without imposing any further conditions on the set function. For example, one could efficiently (linear time in n) compute the scalar and neural Lovász extensions of any set function with only black-box access to the function. This renders our framework more broadly applicable. Furthermore, EGN incorporates the problem constraints additively in the loss function. In contrast to that, our extension framework does not require any commitment to a specific formulation in order to obtain a differentiable loss. This provides more flexibility in modelling the problem, as we can combine the cost function and the constraints in various other ways (e.g., multiplicatively). For general background on neural combinatorial optimization, we refer the reader to the surveys (Bengio et al., 2021; Cappart et al., 2021a; Mazyavkina et al., 2021).

Lifting to high-dimensional spaces. Neural SFEs are heavily inspired by the Goemans-Williamson (Goemans and Williamson, 1995) algorithm and other SDP techniques (Iguchi et al., 2015), which lift problems onto higher dimensional spaces, solve them, and then project back down. Our approach to lifting set functions to high dimensions is motivated by the algorithmic alignment principle (Xu et al., 2019): neural networks whose computations emulate classical algorithms often generalize better with improved sample complexity (Yan et al., 2020; Li et al., 2020; Xu et al., 2019). Emulating algorithmic and logical operations is the focus of Neural Algorithmic Reasoning (Veličković et al., 2019; Dudzik and Veličković, 2022; Deac et al., 2021) and work on knowledge graphs (Hamilton et al., 2018; Ren et al., 2019; Arakelyan et al., 2020), which also emphasize operating in higher dimensions.

Extensions. Scalar SFEs use an LP formulation of the convex closure (El Halabi, 2018, Def. 20), a classical approach for defining convex extensions of discrete functions (Murota, 1998, Eq. 3.57). See Bach (2019) for a study of extensions of submodular functions. The constraints of our dual LP arise in contexts from global optimization (Tawarmalani and Sahinidis, 2002) to barycentric approximation and interpolation schemes in computer graphics (Guessab, 2013;

Hormann, 2014). Convex extensions have also been used for combinatorial penalties with structured sparsity (Obozinski and Bach, 2012, 2016), and general minimization algorithms for set functions (El Halabi and Jegelka, 2020).

Stochastic gradient estimation. SFEs produce gradients for f requiring only black-box access. There is a wide literature on sampling-based approaches to gradient estimation, for instance the REINFORCE algorithm (Williams, 1992) (i.e., score function estimator). However, sampling introduces noise which can cause unstable training and convergence issues, prompting significant study of variance reducing control variates (Gu et al., 2017; Liu et al., 2018; Grathwohl et al., 2018; Wu et al., 2018; Cheng et al., 2020). SFEs can avoid sampling (and noise) all-together, as our extensions are differentiable and can be computed deterministically. A closely related, yet distinct, task is to produce gradients through sampling operations, which introduce non-differentiable nodes in neural network computation graphs. The Straight-Through Estimator (Bengio et al., 2013), arguably the simplest solution, treats sampling as the identity map in the backward pass, yielding biased gradient estimates. The Gumbel-Softmax trick (Maddison et al., 2017; Jang et al., 2017), provides an alternative method to sample from categorical distributions (also benefiting from variance reduction (Paulus et al., 2020b)). The trick can be seen through the lens of the more general Perturb-and-MAP framework that treats sampling as a perturbed optimization program. This framework has since been used to generalize the trick to more complex distributions (Paulus et al., 2020a) and to differentiate through the parameters of exponential families for learning and combinatorial tasks (Niepert et al., 2021). Broadly, these techniques relax a discrete distribution into a continuous one by utilizing a noise distribution and *assuming access* to a continuous loss function. SFEs are complementary to this setup, addressing the problem of designing continuous extensions.

Differentiating through convex programs and algorithms. Recent years have seen a surge of interest in combining neural networks with solvers (e.g., LP solvers) and/or algorithms in differentiable end to end pipelines (Agrawal et al., 2019; Amos and Kolter, 2017; Paulus et al., 2021; Vlastelica et al., 2019; Wang et al., 2019). Whilst sharing the algorithmic alignment motivation of SFEs, the convex programming connection is mostly cosmetic: these works directly embed solvers into network architectures, while SFEs use convex programs as an analytical tool, without requiring solver access.

3.7 Fundamentals of extensions: extended discussion

In this section, we provide an extended discussion of the key components of our LP and SDP formulations and the relationships between them. Apart from supplying derivations, another goal of this section is to illustrate that there is flexibility in the exact choice of formulation for the LP (and consequently the SDP). In fact, since the publication of this work, further connections with literature in convex optimization and semi-definite programming have become apparent. We will delve into them in more detail and discuss some of the tradeoffs when considering which formulation is the appropriate one for defining extensions.

3.7.1 Related linear programs

Our LP formulation depends on a linear program known to correspond to the convex closure (Murota, 1998, Eq. 3.57) (convex envelope) of a discrete function. Some readers may recognize the formal similarities of this formulation with the one used to define the Lovász extension (Bilmes, 2022). Namely, for $\mathbf{x} \in \mathbb{R}^n$ we can define the Lovász Extension as

$$\mathfrak{F}(\mathbf{x}) = \max_{\mathbf{z} \in \mathcal{B}_f} \mathbf{x}^\top \mathbf{z}, \tag{3.3}$$

where the feasible set, known as the base polytope of a submodular function, is defined as $\mathcal{B}_f = \{\mathbf{z} \in \mathbb{R}^n : \mathbf{z}^\top \mathbf{1}_S \leq f(S)\ S \subset [n], \text{ and } \mathbf{z}^\top \mathbf{1}_S = f(S) \text{ when } S = [n]\}$. Base polytopes are also known as *generalized permutahedra* and have rich connections to the theory of matroids, since matroid polytopes belong to the class of generalized permutahedra Ardila et al. (2010).

An alternative option is to consider $\mathbf{x} \in \mathbb{R}^n_+$, then the Lovász extension is given by

$$\mathfrak{F}(\mathbf{x}) = \max_{\mathbf{z} \in \mathcal{P}_f} \mathbf{x}^\top \mathbf{z}, \tag{3.4}$$

where $\mathcal{P}_f$ is the submodular polyhedron as defined in our original primal LP. The subtle differences between those formulations lead to differences in the respective dual formulations. In principle, those formulations can be just as easily used to define set function extensions. Overall, there are three key considerations when defining a suitable LP:

- The constraints of the primal.
- The domain of the primal variables $\mathbf{z}$, b and the cost $\mathbf{x}$.
- The properties of the function being extended.

Below, we describe a few illustrative example cases for different choices of the above:

- Adding the constraint $\mathbf{z}^\top \mathbf{1}_S = f(S)$ when $S = [n]$ leads to $y_{[n]} \in \mathbb{R}^n$ for the dual. This implies that the coefficients cannot be interpreted as probabilities in general which is what provides the guarantee that the extension will not introduce any spurious minima. $\sum_{S \subseteq [n]} y_S = 1$ is just an affine hull constraint in that case.
- For $b = 0$, the constraint $\sum_{S \subseteq [n]} y_S = 1$ is not imposed in the dual and the probabilistic interpretation of the extension cannot be guaranteed. Examples that do not rely on this constraint include the homogeneous convex envelope (El Halabi et al., 2018) and the Lovász extension as presented above. However, even for $b = 0$, from the definition of the Lovász extension it is easy to see that it retains the probabilistic interpretation when $\mathbf{x} \in [0, 1]$.
- Consider a feasible set defined by $\mathcal{P}_f \bigcap \mathbb{R}^n_+$ and let $\mathbf{x} \in \mathbb{R}^n_+$. If the function f is submodular, non-decreasing and normalized so that $f(\varnothing) = 0$ (e.g., the rank function of a matroid),

then the feasible set is called polymatroid and f is a polymatroid function. Again, in that case the Lovász extension achieves the optimal objective value (Schrijver et al., 2003, Eq. 44.32). In that case, the constraint $\sum_{S\subseteq[n]} y_S \mathbf{1}_S = \mathbf{x}$ of the dual is relaxed to $\sum_{S\subseteq[n]} y_S \mathbf{1}_S \geq \mathbf{x}$. This feasible set of the dual will allow for more flexible definitions of an extension but it comes at the cost of generality. For instance, for a submodular function that is not non-decreasing, one cannot obtain the Lovász extension as a feasible solution to the primal LP, and the solutions to this LP will not be the convex envelope in general.

3.7.2 Efficient scalar extensions through matrix inverses.

Historically, the Lovász extension has been one of the most prominent examples of extensions in the literature. It was originally described by Choquet (1954) in the context of the theory of capacities, where it is known as the Choquet integral. Its connections to the greedy algorithm and to submodular functions were made explicit later by Lovász (1983). Indeed, the derivation by Lovász is based on greedily constructing a solution to the optimization problem in 3.4. Here, we will discuss a slightly different perspective on the Lovász extension which facilitates a more general approach to constructing efficient extensions.

Setup. The starting point of this discussion will be the constraint of our dual LP

$$\sum_{i=1}^{n} p_{\mathbf{x}}(S_i)\mathbf{1}_{S_i} = \mathbf{x}. \tag{3.5}$$

Without loss of generality, let the entries of $\mathbf{x} \in [0,1]$ be sorted in non-increasing order. First, we rewrite equation 3.5 in matrix form:

$$\mathbf{S}\mathbf{p} = \mathbf{x}. \tag{3.6}$$

Here we have stacked the supporting set vectors in a matrix $\mathbf{S}$ and their probabilities in a vector $\mathbf{p}$, i.e., $\mathbf{S} = \left[\mathbf{1}_{S_1}, \mathbf{1}_{S_2}, \ldots, \mathbf{1}_{S_n}\right]$ and $\mathbf{p} = \left[p_{\mathbf{x}}(S_1), p_{\mathbf{x}}(S_2), \ldots, p_{\mathbf{x}}(S_n)\right]$. An important consideration when designing extensions is their support. If an extension is supported on a sufficiently small number of sets, then it may be efficiently computed as well. It is worth noting that the number of sets need not be n; any number of supporting sets may be chosen. However, as it will become clear soon, beyond the practical benefits there are also mathematical benefits to selecting n sets for the support of the extension. To construct an extension, we have the following desiderata:

- A binary matrix $\mathbf{S}$ whose columns are the sets that support the extension at point $\mathbf{x}$ and that satisfies equation 3.6.
- Find suitable probabilities $p_{\mathbf{x}}(S_i)$ for each S_i. We should be able to backpropagate through $p_{\mathbf{x}}(S_i)$ to $\mathbf{x}$.

In order for 3.6 to be satisfied for all $\mathbf{x} \in [0,1]^n$, a starting point is to identify a matrix $\mathbf{S}$ that spans $\mathbb{R}^n$. Since we are working with n sets, this implies that the columns of $\mathbf{S}$ have to be linearly independent. It will be instructive to consider a simple example in order to build intuition.

Example (Dual feasible singleton extension)**.** The singletons $S_i = \{i\}$ for $i = 1, 2, \dots, n$ can be used to define a dual feasible extension. Indeed, in that case, the matrix $\mathbf{S}$ is populated by the standard basis vectors $\mathbf{e}_i$ and from equation 3.6 we obtain $\mathbf{Ip} = \mathbf{x}$, where $\mathbf{I}$ is the $n \times n$ identity matrix. As long as we ensure that $\sum_{i=1}^n x_i \leq 1$ holds for $\mathbf{x}$, then $p_{\mathbf{x}}(S_i) = x_i$ is a feasible solution to the dual LP and therefore defines an extension.

The fact that we require n linearly independent vectors for $\mathbf{S}$ also leads to the following observation. $\mathbf{S}$ will be invertible, so we can write

$$\mathbf{Ix} = \mathbf{x}$$
$$\mathbf{S}\underbrace{\mathbf{S}^{-1}\mathbf{x}}_{\mathbf{p_x}} = \mathbf{x}.$$

Therefore, we can cast the problem of finding extensions supported on n sets as the problem of finding $\mathbf{S} \in \{0,1\}^{n\times n}$ such that $\mathbf{x}^\top(\mathbf{S}^\top)^{-1}\mathbf{1} \leq 1$ and the entries of $\mathbf{p}$ are in $[0,1]$, for $\mathbf{x}$ in some suitably chosen subset of $[0,1]^n$. This is both practically appealing and mathematically convenient. We can backpropagate through $\mathbf{S}^{-1}\mathbf{x}$ since it is a linear operation on $\mathbf{x}$. Furthermore, the inverses of several classes of binary matrices are known and have been studied extensively, which simplifies the process of constructing new extensions.

It is clear that the constraints we described above trivially hold for our last singleton example since the identity matrix is its own inverse. In fact, we can also show how the Lovász extension is naturally obtained with this strategy. In the Lovász extension we choose the support sets to be $S_i = \{1, 2, \dots, i\}$ which leads to a binary upper-triangular matrix $\mathbf{S}$ where all the entries above (and including) the main diagonal are ones. It can be shown that the inverse of that binary upper triangular matrix is also upper triangular and it has the following form:

$$\mathbf{S}^{-1}(i,j) = \begin{cases} 1, & j = i \\ -1, & j = i+1 \\ 0, & \text{otherwise.} \end{cases}$$

Then, we can see that the ith entry of $\mathbf{S}^{-1}\mathbf{x}$ is $p_{\mathbf{x}}(S_i) = x_i - x_{i+1}$ which are precisely the coefficients of the Lovász extension.

Constructing novel extensions. The bounded cardinality Lovász extension can be derived because the matrix $\mathbf{S}$ is a banded Toeplitz matrix with a known inverse. Finally, the involutory extension is also supported on singleton sets and relies on the observation that elementary permutation matrices are involutory (i.e., self-inverse) and therefore, as long as the permutation has a fixed point at the maximum element of $\mathbf{x}$, then similar to the singleton example, if $\sum_{i=1}^{n} x_i \leq 1$ we obtain a feasible solution to the dual.

3.7.3 Neural extensions: Intuition and rank one formulation.

In order to motivate the SDP formulation, first we have to identify the essential ingredients of the LP formulation. First, the constraint $\sum_{S\subseteq[n]} y_S \mathbf{1}_S = \mathbf{x}$ captures the simple idea that each continuous point is expressed as a combination of discrete ones, each representing a different set, which is at the core of our extensions. Then, ensuring that the continuous point lies in the convex hull of those discrete points confers additional benefits w.r.t. optimization and offers a probabilistic interpretation.

Consider the following example. The Lovász extension identifies each continuous point in the hypercube with a simplex. Then the continuous point is viewed as an expectation over a distribution supported on the simplex corners. The value of the set function at a continuous point is then the expected value of the function over those corners under the same distribution, i.e., $\mathbb{E}_{S\sim p_{\mathbf{x}}}[\mathbf{1}_S] = \mathbf{x}$ leads to $\mathbb{E}_{S\sim p_{\mathbf{x}}}[f(S)] = \mathfrak{F}(\mathbf{x})$. As long as the distribution $p_{\mathbf{x}}$ can be differentiated w.r.t $\mathbf{x}$, we obtain an extension that can be used with gradient-based optimization. It is clear that the construction depends on being able to identify a small convex set of discrete vectors that can express the continuous one.

This can be formulated in higher dimensions, particularly in the space of PSD matrices. A natural way to represent sets in high dimensions is through rank one matrices that are outer products of the indicator vectors of the sets, i.e., $\mathbf{1}_S\mathbf{1}_S^\top$ is the matrix representation of S similar to how $\mathbf{1}_S$ is the vector representation. Hence, in the space of matrices, our goal will be again to identify a PSD matrix with linear combinations of *matrices* that represent sets. This is done through an expansion of the PSD matrix as a convex combination of outer products between set indicator vectors, which we obtain via the eigendecomposition of the PSD matrix.

The above considerations set the stage for a transition from linear programming to semidefinite programming, where the feasible sets are spectrahedra. Our SDP formulation attempts to capture the intuition described in the previous paragraphs while also maintaining formal connections to the LP by showing that feasible LP regions correspond to feasible SDP regions by simply projecting the LP regions on the space of diagonal matrices (see Proposition 2).

Rank one formulation. The SDP formulation we have described so far maps terms of the form $\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top$ to evaluations $f(S\cap T)$. When $S = T$, this maps outer products (up to suitable rescaling) to $f(S\cap S) = f(T) = f(S)$, which secures the extension property. The convention of using rank two matrices happens to be mathematically convenient and practically viable. However, it can be argued that those rank two matrices do not represent set intersections naturally; a standard way to represent set intersections with matrices would be to use rank one matrices $\mathbf{1}_{S\cap T}\mathbf{1}_{S\cap T}^\top$. The question then is whether we can reformulate our SDP in terms of rank one matrices so that the additional rank two correspondence with intersections is not imposed. The challenge is to achieve this while also ensuring that feasible solutions to the corresponding dual can give rise to extensions in the same fashion as our current formulation.

As we will show, this is actually possible. We begin with the primal SDP.

$$\max_{\mathbf{Z}\succeq 0, b\in\mathbb{R}} \{\mathrm{Tr}(\mathbf{X}^\top \mathbf{Z}) + b\} \text{ subject to } \mathbf{1}_S^\top \mathbf{Z} \mathbf{1}_S + b \leq f(S) \text{ for all } S \subseteq [n]. \qquad \text{(rank-one primal SDP)}$$

Following standard conversion rules we obtain the dual

$$\min_{\{y_S \geq 0\}} \sum_{S\subseteq[n]} y_S f(S) \text{ subject to } \mathbf{X} \preceq \sum_{S\subseteq[n]} y_S \mathbf{1}_S \mathbf{1}_S^\top \text{ and } \sum_{S\subseteq[n]} y_S = 1. \qquad \text{(rank-one dual SDP)}$$

Proposition 4. The rank-one dual SDP is feasible.

Proof. The proof strategy that we follow is similar to the one provided for neural extensions, with an additional step that relies on the following lemma.

Lemma 1. Let V be a set of n elements. Let $S_i, S_j \subseteq V$, and $\mathbf{1}_{S_i}, \mathbf{1}_{S_j}$ be their corresponding characteristic vectors. We have

$$\mathbf{1}_{S_i}\mathbf{1}_{S_j}^\top + \mathbf{1}_{S_j}\mathbf{1}_{S_i}^\top = \mathbf{1}_{U_{ij}}\mathbf{1}_{U_{ij}}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top - (\mathbf{1}_{S_j\setminus I_{ij}})(\mathbf{1}_{S_j\setminus I_{ij}})^\top - (\mathbf{1}_{S_i\setminus I_{ij}})(\mathbf{1}_{S_i\setminus I_{ij}})^\top, \tag{3.7}$$

where $U_{ij} = S_i \bigcup S_j, I_{ij} = S_i \bigcap S_j$.

See B.2.2 for proof.

To complete the proof of the proposition, consider a decomposition $\mathbf{X} = \sum_{i=1}^n \lambda_i \mathbf{x}_i \mathbf{x}_i^\top$ with $\mathbf{x}_i \in [0,1]^n$ and $\sum_{i=1}^n \lambda_i = 1$. Also, let U_{ST} be the union of sets S and T, and I_{ST} their intersection. Let $\mathbf{x}_i = \sum_S p_{\mathbf{x}_i}(S)$ be the coefficients of a feasible scalar SFE. Then,

$$\begin{aligned}
\mathbf{x}_i\mathbf{x}_i^\top &= \big(\sum_S p_{\mathbf{x}_i}(S)\mathbf{1}_S\big)\big(\sum_T p_{\mathbf{x}_i}(T)\mathbf{1}_T\big)^\top \\
&= \sum_S p_{\mathbf{x}_i}(S)^2 \mathbf{1}_S\mathbf{1}_S^\top + \sum_{S\neq T} p_{\mathbf{x}_i}(S)p_{\mathbf{x}_i}(T)(\mathbf{1}_T\mathbf{1}_S^\top + \mathbf{1}_S\mathbf{1}_T^\top) \\
&\overset{3.7}{=} \sum_S p_{\mathbf{x}_i}(S)^2 \mathbf{1}_S\mathbf{1}_S^\top + \sum_{S\neq T} p_{\mathbf{x}_i}(S)p_{\mathbf{x}_i}(T)(\mathbf{1}_{U_{ST}}\mathbf{1}_{U_{ST}}^\top + \mathbf{1}_{I_{ST}}\mathbf{1}_{I_{ST}}^\top - \mathbf{1}_{S\setminus I_{ST}}\mathbf{1}_{S\setminus I_{ST}}^\top - \mathbf{1}_{T\setminus I_{ST}}\mathbf{1}_{T\setminus I_{ST}}^\top) \\
&\preceq \sum_S p_{\mathbf{x}_i}(S)^2 \mathbf{1}_S\mathbf{1}_S^\top + \sum_{S\neq T} p_{\mathbf{x}_i}(S)p_{\mathbf{x}_i}(T)(\mathbf{1}_{U_{ST}}\mathbf{1}_{U_{ST}}^\top + \mathbf{1}_{I_{ST}}\mathbf{1}_{I_{ST}}^\top).
\end{aligned} \tag{3.8}$$

Thus, the LMI in the constraint is being satisfied. We can also verify that the constraint $\sum_{S\subseteq[n]} y_S = 1$ holds because $\sum_{i=1}^n \lambda_i \big(\sum_S p_{\mathbf{x}_i}(S)^2 + 2\sum_{S\neq T} p_{\mathbf{x}_i}(S)p_{\mathbf{x}_i}(T)\big) = 1$. □

It should be apparent from the proof technique that we may build neural extensions based on any feasible scalar ones in this setting again. Now each rank-one matrix that corresponds to a set S is paired to a corresponding function evaluation $f(S)$. Another difference that emerges, in this case, is that for the properties of the scalar extension sets to be maintained in the neural extension, they would have to be closed both under union and intersection. For example, a neural extension that is based on a cardinality-constrained scalar extension will not preserve the cardinality constraint. This issue may be alleviated by carefully tweaking 3.8 to replace the union terms while preserving the LMI so that a feasible solution is achieved.

Sum of squares and related optimization programs. An interesting aspect of this version of the optimization program is its formal similarity with SDPs that can be formulated for function minimization using the sum of squares (SOS) hierarchy (Lasserre, 2001). Since **Z** is a PSD matrix, we can see that the constraints of the rank-one primal SDP can be written as

$$\sum_{i=1}^{n} \lambda_i (\mathbf{1}_S^\top \mathbf{x_i})^2 \leq f(S) - b \quad S \subseteq [n].$$

In other words, the function is lower bounded by a *sum of squares*. For example, in binary optimization with SOS over the Boolean hypercube, one typically encounters the following formulation (Slot and Laurent, 2023) for a tractable lower bound on the function f

$$f_r := \quad \max b \quad \text{subject to} \quad f(\mathbf{1}_S) - b = g(\mathbf{1}_S), \text{ for all } \mathbf{1}_S \in \{0,1\}^n,$$

where $g(x)$ is a sum of squares polynomial of degree at most $2r$. The sum of squares constraint enables framing this as a semi-definite program and therefore makes it solvable in polynomial time.

Another work that shares formal similarities with the SOS approach and our rank-one primal SDP formulation is the one in Rudi et al. (2020). There, the function being minimized is defined over $\mathbb{R}^d$ and a regularization term is added to the objective, so $b - \alpha\mathrm{Tr}(\mathbf{Z})$ is being maximized, for some PSD matrix **Z** and some positive coefficient α. This is particularly close to the SDP we have presented, as the **Z** in that paper is also decomposed as a product of two factors. However, in contrast to our treatment, the product in Rudi et al. (2020) allows for controlled function approximation using kernels and through subsampling of the SOS-style constraints at a few points in order to formulate the primal SDP.

3.7.4 Generalizing extensions to different geometries.

Conic programming. Due to the impact that semi-definite programming has had in the field of combinatorial optimization (Lovász, 2003; Goemans, 1997), the choice of the cone of PSD matrices is a rather natural one for our extensions. On the other hand, it is interesting to test whether the main ideas behind extensions can be generalized to other geometries. The motivation behind this is twofold. In terms of formal limitations, it has been shown that there are compact convex semi-algebraic sets (i.e., convex sets that can be written as a finite union of sets defined by polynomial equalities and inequalities) that are not representable by a semi-definite program. As an example, minimizing a polynomial of $n \geq 3$ variables and degree at least 4 over the unit ball in $\mathbf{R}^n$ cannot be modeled exactly by a semi-definite program (Scheiderer, 2018). There are also practical considerations that motivate us to pursue formulations beyond the PSD cone. Recall that an important element of our approach is to obtain a feasible solution to the dual SDP through a suitable decomposition of $\mathbf{X}$ in terms of vectors $\mathbf{x_i} \in [0,1]^n$. Clearly, not all PSD matrices can guarantee this decomposition so we construct it by tweaking the eigendecomposition of the Gram matrix. However, this decomposition is captured accurately by the cone of *completely positive matrices*. We will shortly see how we can define an optimization program that will make this requirement explicit.

First, we provide some background information about conic programming (Vandenberghe and Boyd, 2004), which will help us formalize a more general approach. Let K be a cone. The set

$$K^* = \{y | \langle z, y\rangle \geq 0 \text{ for all } z \in K\}$$

is called the dual cone of K. Here, $\langle\cdot,\cdot\rangle$ denotes the standard Euclidean inner product. The cone of PSD matrices, is known to be self-dual, i.e., its dual is also the cone of PSD matrices. In conic programming, duality dictates that if the primal inequalities are over the cone K, then the dual inequalities will be over its dual K^*.

Therefore, a general convex optimization program for extensions over arbitrary cones, given suitable embeddings $e(S)$ into the cone, may be written as

$$\min_{\{y_S \geq 0\}} \sum_{S\subseteq[n]} y_S f(S) \text{ subject to } \sum_{S\subseteq[n]} y_S e(S) - \mathbf{X} \in K^* \text{ and } \sum_{S\subseteq[n]} y_S = 1. \tag{3.9}$$

Connections to copositive programming. We now return to our earlier consideration regarding completely positive matrices. The cone of completely positive matrices is defined as

$$\mathcal{C}^* = \operatorname{conv}\{\mathbf{x}\mathbf{x}^\top : \mathbf{x} \in \mathbb{R}^n_+\}.$$

The dual of the completely positive matrix cone $(\mathcal{C}^*)^* = \mathcal{C}$ is called the cone of *copositive matrices* and is defined as

$$\mathcal{C} = \{\mathbf{Z} \in \mathbb{S}^n : \mathbf{x}^\top \mathbf{Z}\mathbf{x} \geq 0, \mathbf{x} \in \mathbb{R}^n_+\}.$$

Similarly, because both cones are closed convex cones, by taking the dual of the cone of copositive matrices we obtain the cone of completely positive matrices. The area of copositive programming has been developed to study optimization programs formulated over the copositive cone and its dual. Copositive programming has a long history in combinatorial optimization and famous problems like the maximum clique and maximum independent set may be formulated exactly as copositive programs. Copositive programs have several interesting properties. While they are convex optimization problems, they can still be NP-Hard. Since they do not admit efficiently computable self-concordant barrier functions, they cannot be solved with interior point methods (Dür, 2010). Deciding whether a matrix belongs to the cone of completely positive matrices is also known to be NP-Hard (Dickinson and Gijben, 2014). Furthermore, it has recently been shown that the cones of copositive matrices of size more than 5 cannot be represented as spectrahedral shadows (i.e., feasible regions of SDPs) (Bodirsky et al., 2022).

Thankfully, those challenging aspects of copositive programming will not be of issue in our approach, since we do not need to solve the optimization problem. As long as we can efficiently construct completely positive matrices within our neural network pipeline in a differentiable way, we will be able to construct neural extensions. Using the general formulation for cone programming we described above, we may define the following optimization problem

$$\max_{\mathbf{Z}\in\mathcal{C},b\in\mathbb{R}} \{\mathrm{Tr}(\mathbf{X}^\top \mathbf{Z}) + b\} \text{ subject to } \mathbf{1}_S^\top \mathbf{Z}\mathbf{1}_S + b \leq f(S) \text{ for } S \subseteq [n]. \qquad \text{(copositive primal)}$$

$$\min_{\{y_S\geq 0\}} \sum_{S\subseteq[n]} y_S f(S) \text{ subject to } \sum_{S\subseteq[n]} y_S \mathbf{1}_S \mathbf{1}_S^\top - \mathbf{X} \in \mathcal{C}^* \text{ and } \sum_{S\subseteq[n]} y_S = 1. \qquad \text{(copositive dual)}$$

Proposition 5. The copositive dual is feasible.

The proof of this proposition is identical to the feasibility proof for the rank-one dual SDP. Feasible solutions of this version of the optimization problem provide a more natural definition for extensions that precisely match the proof technique. Since our construction from the proof of Proposition 4 requires $\mathbf{X} = \sum_{i=1}^n \lambda_i \mathbf{x}_i \mathbf{x}_i^\top$ with $\mathbf{x}_i \in [0,1]^n$ and $\sum_{i=1}^n \lambda_i = 1$, this is equivalent to requiring that $\mathbf{X}$ is a suitably normalized completely positive matrix, which is what is imposed now by the constraints of the copositive dual.

Alternative formulations in the space of matrices. So far, we have provided concrete examples of high-dimensional extensions on cones of symmetric matrices. However, there are problems where modeling via neural extensions on the cones of symmetric matrices may not be convenient. Consider the task of finding the optimal tour for the Euclidean TSP. Our approaches so far have been designed for the purpose of learning to find sets as solutions to combinatorial problems. On the other hand, an important ingredient of this TSP problem is predicting the correct order of a set of nodes, which can be elegantly captured by predicting a permutation matrix.

We will discuss how we may define an extension in the space of permutation matrices for the purpose of solving routing problems like the euclidean TSP. We start by setting up the following primal problem. Let $\mathcal{P}^n$ be the set of all $n \times n$ permutation matrices. For an input doubly stochastic matrix $\mathbf{X} \in [0,1]^{n\times n}$ and a function over permutations $f : \mathcal{P}^n \to \mathbb{R}$, we have

$$\max_{\mathbf{Z}\in\mathbb{R}^{n\times n}, b\in\mathbb{R}} \{\mathrm{Tr}(\mathbf{X}^\top \mathbf{Z}) + b\} \text{ subject to } \mathrm{Tr}(\mathbf{P}^\top \mathbf{Z}) + b \leq f(\mathbf{P}) \text{ for all } \mathbf{P} \in \mathcal{P}^n. \quad \text{(permutation primal LP)}$$

Duality can be carried out in the same way in matrix variable linear programs (Craven and Mond, 1981). Therefore, we may write the following dual:

$$\min_{\{y_\mathbf{P}\geq 0\}} \sum_{\mathbf{P}\in\mathcal{P}^n} y_\mathbf{P} f(\mathbf{P}) \text{ subject to } \mathbf{X} = \sum_{\mathbf{P}\in\mathcal{P}^n} y_\mathbf{P}\mathbf{P}, \text{ and } \sum_{\mathbf{P}\in\mathcal{P}^n} y_\mathbf{P} = 1. \qquad \text{(permutation dual LP)}$$

Proposition 6. The permutation dual LP is feasible.

Proof. Since $\mathbf{X}$ is doubly stochastic, the Birkhoff-von Neumann theorem (Birkhoff, 1946; Von Neumann, 1953) states that it is in the convex hull of all permutation matrices of the same size. Therefore, there exist $a_i \in [0,1]$ and corresponding permutation matrices $\mathbf{P}_i$ such that $\mathbf{X} = \sum_i a_i \mathbf{P}_i$, $\sum_i a_i = 1$. Thus, by taking $a_i = y_{\mathbf{P}_i}$ we obtain a feasible solution. □

In this formulation, the decomposition into permutation matrices consists of $O(n^2)$ terms in general. Conveniently, it can be efficiently approximated with coefficients that are differentiable functions of the entries of $\mathbf{X}$ (Dufossé and Uçar, 2016). This allows us to define extensions of functions whose domain consists of the extremal points of the Birkhoff polytope. Finally, to satisfy the requirement that $\mathbf{X}$ is a doubly stochastic matrix, we may employ Sinkhorn's algorithm (Sinkhorn, 1964) to convert a matrix of neural network embeddings to a doubly stochastic matrix. This has been extensively used in the machine learning literature in the context of learning permutations (Mena et al., 2018; Emami and Ranka, 2018).

3.8 Optimization aspects of extensions

In this section, we will discuss the minima and continuity properties of extensions that can be critical to their success in the context of solving tasks with neural networks. In this process, we will also (re)-state and prove some results for SFEs.

3.8.1 The minima of extensions

The first result concerns the minima of $\mathfrak{F}$, showing that the minimum value is the same as that of f, and no additional minima are added (besides convex combinations of discrete minimizers). These properties are especially desirable when using an extension $\mathfrak{F}$ as a loss function (see Section 3.5) since it is important that $\mathfrak{F}$ drive the neural network NN_1 towards producing discrete $\mathbf{1}_S$ outputs.

Proposition 7 (Scalar SFEs have no bad minima)**.** If $\mathfrak{F}$ is a scalar SFE of f then:

1. $\min_{\mathbf{x}\in\mathcal{X}} \mathfrak{F}(\mathbf{x}) = \min_{S\subseteq[n]} f(S)$

2. $\operatorname{argmin}_{\mathbf{x}\in\mathcal{X}} \mathfrak{F}(\mathbf{x}) \subseteq \text{Hull}\big(\operatorname{argmin}_{\mathbf{1}_S : S\subseteq[n]} f(S)\big)$

Proof. The inequality $\min_{\mathbf{x}\in\mathcal{X}} \mathfrak{F}(\mathbf{x}) \leq \min_{S\subseteq[n]} f(S)$ automatically holds since $\min_{S\subseteq[n]} f(S) = \min_{\mathbf{1}_S : S\subseteq[n]} \mathfrak{F}(\mathbf{1}_S)$, and $\{\mathbf{1}_S : S \subseteq [n]\} \subseteq \mathcal{X}$. So it remains to show the reverse. Indeed, letting $\mathbf{x} \in \mathcal{X}$ be an arbitrary point we have,

$$\begin{aligned}\mathfrak{F}(\mathbf{x}) &= \mathbb{E}_{S\sim p_{\mathbf{x}}}[f(S)] \\ &\geq \sum_{S\subseteq[n]} p_{\mathbf{x}}(S) \cdot \min_{S\subseteq[n]} f(S) \\ &= \min_{S\subseteq[n]} f(S)\end{aligned}$$

where the last equality simply uses the fact that $\sum_{S\subseteq[n]} p_{\mathbf{x}}(S) = 1$. This proves the first claim.

To prove the second claim, suppose that $\mathbf{x}$ minimizes $\mathfrak{F}(\mathbf{x})$ over $\mathbf{x} \in \mathcal{X}$. This implies that the inequality in the above derivation must be tight, which is true if and only if

$$p_{\mathbf{x}}(S) \cdot f(S) = p_{\mathbf{x}}(S) \cdot \min_{S\subseteq[n]} f(S) \quad \text{for all } S \subseteq [n].$$

For a given S, this implies that either $p_{\mathbf{x}}(S) = 0$ or $f(S) = \min_{S\subseteq[n]} f(S)$. Since $\mathbf{x} = \mathbb{E}_{S\sim p_{\mathbf{x}}}[\mathbf{1}_S] = \sum_{S\subseteq[n]} p_{\mathbf{x}}(S) \cdot \mathbf{1}_S = \sum_{S:p_{\mathbf{x}}(S)>0} p_{\mathbf{x}}(S) \cdot \mathbf{1}_S$. This is precisely a convex combination of points $\mathbf{1}_S$ for which $f(S) = \min_{S\subseteq[n]} f(S)$. Since $\mathfrak{F}$ is a convex combination of exactly this set of points $\mathbf{1}_S$, we have the second claim.

□

3.8.2 Smoothness and continuity of extensions

It is useful to identify conditions that guarantee that an extension will be “well-behaved” in practice when used in a neural network pipeline. Certainly, one of the conditions that must be met is that $\mathfrak{F}$ can be used with automatic differentiation that is routinely done in machine learning packages. Almost everywhere differentiability is generally desirable in that context, as in practice, the existence of a measure-zero set of non-differentiabilities (e.g., ReLUs) does not seem to be impactful in modern learning pipelines. Indeed, the extensions we have presented are piece-wise linear, which guarantees that they are almost everywhere differentiable. On the other hand, the continuity properties of the extensions are slightly more complicated. A sufficient condition for continuity (and almost everywhere differentiability) is to show that $\mathfrak{F}$ is Lipschitz. A straightforward calculation demonstrates that it suffices to show that $\mathbf{x} \in \mathcal{X} \mapsto p_{\mathbf{x}}(S)$ is Lipschitz continuous.

Lemma 2. If the mapping $\mathbf{x} \in [0,1]^n \mapsto p_{\mathbf{x}}(S)$ is Lipschitz continuous and $f(S)$ is finite for all S in the support of $p_{\mathbf{x}}$, then $\mathfrak{F}$ is also Lipschitz continuous. In particular, $\mathfrak{F}$ is continuous and almost everywhere differentiable.

Proof. The Lipschitz continuity of $\mathfrak{F}(\mathbf{x})$ follows directly from the definition:

$$\begin{aligned}\left|\mathfrak{F}(\mathbf{x})-\mathfrak{F}(\mathbf{x}')\right| &= \left|\sum_{S\subseteq[n]} p_{\mathbf{x}}(S)\cdot f(S)-\sum_{S\subseteq[n]} p_{\mathbf{x}'}(S)\cdot f(S)\right| \\ &= \left|\sum_{S\subseteq[n]} \big(p_{\mathbf{x}}(S)-p_{\mathbf{x}'}(S)\big)\cdot f(S)\right| \le \left(2kL\max_{S\subseteq[n]} f(S)\right)\cdot\|\mathbf{x}-\mathbf{x}'\|,\end{aligned}$$

where L is the maximum Lipschitz constant of $\mathbf{x}\mapsto p_{\mathbf{x}}(S)$ over any S in the support of $p_{\mathbf{x}}$, and k is the maximal cardinality of the support of any $p_{\mathbf{x}}$. □

In general, k can be trivially bounded by 2^n, so $\mathfrak{F}$ is always Lipschitz. However in many cases the cardinality of the support of any $p_{\mathbf{x}}$ is much smaller than 2^n, leading to a smaller Lipschitz constant. For instance, $k=n$ in the case of the Lovász extension.

In light of this observation, it is worth examining whether Lipschitzness can be guaranteed in our approach to constructing extensions that we presented in 3.7.2. Recall that our strategy hinges on the ability to find a binary square matrix of supporting sets whose inverse maps the input vector $\mathbf{x}$ to a probability distribution.

In this context, it is useful to consider the comparison between the Lovász extension and its bounded cardinality version. Based on the proof of Lemma 3 of the Lipschitzness of the Lovász extension, we can make the following observation. Due to piecewise linearity, the extension will be Lipschitz at each piece. On the other hand, depending on how the supporting sets S are chosen, we may have discontinuities at the boundaries between pieces. This means that we cannot guarantee that the extension generated by our construction will be Lipschitz without imposing additional conditions on how the support of $\mathfrak{F}$ varies between pieces (i.e., how changes in the coordinate ranking of $\mathbf{x}$ lead to changes in $\mathbf{S}$).

It should also be noted that recent work has shown that a condition stronger than almost everywhere differentiability called piecewise analyticity under analytic partition (PAP) can guarantee the formal correctness of modern automatic differentiation packages when those are applied to functions with non-differentiabilities (Lee et al., 2020). Investigating the PAP conditions in the context of SFEs to obtain formal guarantees of correctness is an interesting future direction.

3.9 Revisiting the probabilistic method

In the previous chapter, subsection 2.7.2 discussed two of the key limitations of the probabilistic method for deriving differentiable losses using the product measure. Briefly summarized, the two unaddressed limitations from that chapter were

- learning with black box objectives and constraints
- differentiating through complicated expressions for the objective or the probability of

constraint violation $P(S \notin \Omega)$.

Many of the SFEs we have discussed require only black-box access to the function so the first bullet point is immediately addressed. The second bullet point concerns the difficulty of obtaining differentiable closed-form expressions for complicated objectives and constraints, even when their explicit description is available.

Recall that the probabilistic penalty loss for some nonnegative cost function f, a sufficiently large penalty coefficient $\beta \in \mathbb{R}^+$, and a graph $G = (V, E)$ is

$$\ell(\mathcal{D}; G) = \mathbb{E}\left[f(S; G)\right] + \beta P(S \notin \Omega).$$

As we have already mentioned earlier on, our construction from the previous chapter can be viewed as a special case of a multilinear extension. Indeed, theorem 3 from the previous chapter relies on the fact that the probabilistic penalty loss is an extension of the cost function on the feasible set.

The challenge in the second bullet point originates in the choice of distribution for the extension. Indeed by abandoning the product measure, we can use extensions supported on a smaller number of sets (e.g., n sets for the Lovász extension). This in turn allows us to easily compute the term $\mathbb{E}\left[f(S; G)\right]$ for any cost function. For $P(S \notin \Omega)$, we may use any (suitably normalized) function $g(S; G) : 2^n \to \mathbb{R}^+$ that measures constraint violation. Then, from Markov's inequality we can straightforwardly bound $P(S \notin \Omega)$ with $\mathbb{E}\left[g(S; G)\right]$ by computing the extension of g. This leads to the following extension-based probabilistic penalty loss

$$\ell(\mathcal{D}; G)_{\text{sfe}} = \mathbb{E}\left[f(S; G)\right] + \beta \mathbb{E}\left[g(S; G)\right].$$

As long as $g(S; G) = 0$ if and only if $S \in \Omega$, this modified version of the loss can be efficiently computed for all functions, even in a black box setting, while also maintaining the properties of the probabilistic penalty loss. Specifically, we can prove that the minima of this loss match the minima of the constrained optimization problem.

Proposition 8. $\ell(\mathcal{D}; G)_{\text{sfe}}$ has no bad minima.

Proof. For $\mathcal{D}$ supported only on $S \in \Omega$, $\ell(\mathcal{D}; G)_{\text{sfe}} = \mathbb{E}\left[f(S; G)\right]$ and is therefore an extension. Proposition 7 then implies that $\ell(\mathcal{D}; G)_{\text{sfe}}$ has the same minima as f over the feasible set. If $\mathcal{D}$ supports infeasible solutions then

$$\ell(\mathcal{D}; G)_{\text{sfe}} \geq \min_{S \in \Omega} f(S; G) + \beta \mathbb{E}\left[g(S; G)\right] + \sum_{S \notin \Omega} P(S) f(S; G).$$

We know that $\beta \mathbb{E}\left[g(S; G)\right] + \sum_{S \notin \Omega} P(S) f(S; G) > 0$ because $\beta > 0$ and $\mathbb{E}\left[g(S; G)\right] > 0$ (g is 0 only at feasible solutions) which completes the proof. □

One of the primary limitations of the probabilistic method was the use of the product distribution. Using extensions supported on different distributions as we have described in this chapter circumvents some of the central limitations of the approach. Nevertheless, there are tradeoffs to consider in this context. While other distributions allow us to tractably compute expectations, this may come at a cost of sufficiently exploring an exponentially large space. Indeed, carefully considering the choice of extension depending on prior knowledge about the task will be crucial in successfully applying extensions in realistic settings.

3.10 Conclusion

We introduced Neural Set Function Extensions, a framework that enables evaluating set functions on continuous and high-dimensional representations. We showed how to define such extensions so that they can be used in end-to-end differentiable models and demonstrated their viability in a range of tasks including combinatorial optimization and image classification. Notably, neural extensions deliver good results and improve over their scalar counterparts, further affirming the benefits of problem-solving in high dimensions.

4 Conclusion

4.1 Summary

In this thesis, we have designed tools that enable the use of discrete functions in end-to-end differentiable models, and that can train neural networks to solve combinatorial optimization problems without supervision.

To achieve this, we leveraged the probabilistic method to construct differentiable loss functions that can be used for unsupervised training. We were able to show that by minimizing those loss functions the network learns a distribution that provably contains high-quality feasible solutions. The proof of the existence of those solutions was then derandomized using the method of conditional expectation. This provided us with a deterministic way to decode high-quality solutions out of the neural network outputs at inference time. Empirically, this method has established the state of the art in unsupervised combinatorial optimization with neural networks. It was evaluated on a series of datasets involving real-world instances and hard synthetic problems. For the case of problems with linear box constraints, we provided an additional iterative algorithm that updates the learned distribution until the constraints are satisfied. Moreover, we demonstrated how we can derive probabilistic penalty loss functions for several landmark problems in combinatorial optimization like the maximum independent set, CNF-SAT, and minimum dominating set. We then provided an extended discussion of the limitations in the proposed methodology. Finally, we explained how more powerful tools of the probabilistic method like the Lovász local lemma can be used with ML models to overcome some of those limitations by exploiting the sparsity of instances to obtain existence proofs of constraint satisfiability. Furthermore, we described how the proofs can be made constructive and practical via the Moser-Tardos algorithm.

In order to tackle some of the limitations of the approach in Chapter 2 and to provide a method for using discrete set functions in differentiable models we described a general framework for continuous extensions. Through the linear programming formulation of the convex closure, we defined scalar set function extensions which extend the domain of discrete functions from $\{0,1\}^n$ to the entire hypercube $[0,1]^n$. Our extensions have a probabilistic interpretation since they can be viewed as efficiently computable expectations over function evaluations on the corners of the hypercube. We showed that our extensions do not introduce bad minima and can be used effectively in end-to-end differentiable neural network pipelines. The versatility of our approach extends beyond combinatorial optimization as we were able to train a competitive image classifier through an extension of the training error function. In order to facilitate the discovery of new extensions, we described a methodology for constructing extensions through matrix inverses that can be used to derive known extensions like the Lovász extension.

The same methodology allowed us to create a cardinality-constrained generalization of the Lovász extension that is supported on sets of cardinality at most k. The use of this extension led to improved experimental results for constraint satisfaction problems. In order to tackle the problem of discrete computation in high dimensions, we proposed defining extensions on the cone of positive semidefinite matrices via a suitable semidefinite program and showed how extensions may be built there on top of scalar extensions. Extensions defined on higher dimensional domains improved over their lower dimensional counterparts and outperformed REINFORCE on combinatorial optimization problems without any significant computational overhead. We also discussed how to define extensions on other higher dimensional geometries like the cone of completely positive matrices and the Birkhoff polytope. Finally, we showed how the formalism of extensions can be used to improve probabilistic penalty losses by making the expectation of any constraint function tractable, which in turn enables the use of the probabilistic penalty loss even for complex or black-box constraints.

4.2 Future directions

There are several directions that may be explored in order to augment and expand the methodologies we have described. At a finer-grained level, exploring ways to include concentration inequalities in the probabilistic penalty loss in a differentiable way could lead to better control of the properties of the learned distribution. Moreover, it could secure stronger guarantees for the solutions that are being recovered. Another avenue that requires further exploration is the derandomization procedure. The method of conditional expectation requires recalculating expected values after each discretization step which can be costly (e.g., for the product measure). Developing faster variants or stopping criteria, e.g., via pessimistic estimators (Alon and Spencer, 2016), could significantly improve the practical appeal of the method.

Regarding extensions, even though we have discussed general guidelines on how to generate new extensions under specific conditions, those guidelines are by no means exhaustive. The methodology described in 3.7.2 is inherently limited to n supporting sets. Ideally, we would like to be able to control the tradeoff between efficiency and exploration by varying the number of sets in the support of the extension, while also accommodating sets with more complex constraints. Furthermore, providing a consistent design methodology while ensuring properties like Lipschitz continuity remains an open problem. Beyond the use case of extensions as loss functions that we have demonstrated in our experiments, an interesting prospect is to explore extensions for neural algorithmic reasoning (Velickovic et al., 2020) and more broadly as a tool in the design of differentiable algorithms (Petersen, 2022). In that context, extensions may provide differentiable substitutes for the internal operations of algorithms. Alternatively, extensions could also serve as powerful extractors of structural features (e.g., structural properties of induced subgraphs) which could be leveraged to increase the expressivity of neural network architectures (Chen and Tian, 2019; Bouritsas et al., 2022). Finally, a direction that merits further investigation involves building problem symmetries into our extensions. In that vein, it would be interesting to thoroughly study and build (neural) extensions on convex bodies generated from group representations, which in certain cases admit spectrahedral representations (Sanyal et al., 2011; Saunderson et al., 2015).

A Appendix of Chapter 2

A.1 Proofs and Experimental Details

A.1.1 Proof of Theorem 1

Theorem 1. Fix any $\beta > \max_S f(S;G)$ and let $\ell(\mathcal{D};G) < (1-t)\beta$. With probability at least t, set $S^* \sim \mathcal{D}$ satisfies

$$f(S^*;G) < \ell(\mathcal{D};G)/(1-t) \text{ and } S^* \in \Omega,$$

under the condition that f is non-negative.

Proof. In the constrained case, the focus is on the probability $P(\{f_{(S;G)} < \epsilon\} \cap \{S \in \Omega\})$. Define the following probabilistic penalty function:

$$f_p(S;G) = f(S;G) + \mathbf{1}_{S \notin \Omega}\,\beta, \tag{A.1}$$

where β is any number larger than $\max_S\{f(S;G)\}$. The key observation is that, if $\ell(\mathcal{D},G) = \epsilon < \beta$, then there must exist a valid solution of cost ϵ. It is a consequence of $f(S;G) > 0$ and β being an upper bound of f that

$$P(f_p(S;G) < \epsilon) = P(f(S;G) < \epsilon \cap S \in \Omega). \tag{A.2}$$

Similar to the unconstrained case, for a non-negative f, Markov's inequality can be utilized to bound this probability:

$$\begin{aligned}
P(\{f(S;G) < \epsilon\} \cap \{S \in \Omega\}) &= P(f_p(S;G) < \epsilon) \\
&> 1 - \frac{1}{\epsilon}\mathbb{E}\left[f_p(S;G)\right] \\
&= 1 - \frac{1}{\epsilon}\left(\mathbb{E}\left[f(S;G)\right] + \mathbb{E}\left[\mathbf{1}_{S \notin \Omega}\,\beta\right]\right) \\
&= 1 - \frac{1}{\epsilon}\left(\mathbb{E}\left[f(S;G)\right] + P(S \notin \Omega)\,\beta\right).
\end{aligned} \tag{A.3}$$

The theorem claim follows from the final inequality. □

A.1.2 Proof of Theorem 2

Theorem 2. Let $\mathcal{D}$ be the distribution obtained after successful re-scaling of the probabilities. For any (unconstrained) probabilistic loss function that abides to $P(f(S;G) < \ell(\mathcal{D};G)) > t$, set $S^* \sim \mathcal{D}$ satisfies $f(S^*;G) < \ell(\mathcal{D};G)$ and $\sum_{v_i \in S^*} a_i \in [b_l, b_h]$, with probability at least

$t - 2\exp\left(-(b_h - b_l)^2 / \sum_i 2a_i^2\right)$.

Proof. Set $b = (b_l + b_h)/2$ and $\delta = (b_h - b_l)/2$. By Hoeffding's inequality, the probability that a sample of $\mathcal{D}$ will lie in the correct interval is:

$$P\left(\left|\sum_{v_i \in S} a_i - \mathbb{E}\left[\sum_{v_i \in S} a_i\right]\right| \leq \delta\right) = P\left(\left|\sum_{v_i \in S} a_i - b\right| \leq \delta\right) \geq 1 - 2\exp\left(-\frac{2\delta^2}{\sum_i a_i^2}\right).$$

We can combine this guarantee with the unconstrained guarantee by taking a union bound over the two events:

$$\begin{aligned} P&\left(f(S;G) < \ell(\mathcal{D},G) \text{ AND } \sum_{v_i \in S} a_i \in [b_l, b_h]\right) \\ &= 1 - P\left(f(S;G) \geq \ell(\mathcal{D},G) \text{ OR } \sum_{v_i \in S} a_i \notin [b_l, b_h]\right) \\ &\geq 1 - P\left(f(S;G) \geq \ell(\mathcal{D},G)\right) - P\left(\sum_{v_i \in S} a_i \notin [b_l, b_h]\right) \\ &\geq t - 2\exp\left(-\frac{2\delta^2}{\sum_i a_i^2}\right) \end{aligned}$$

The previous is positive whenever $t > 2\exp\left(-2\delta^2 / (\sum_i a_i^2)\right)$. □

Proof of Corollary 1

Corollary 1. Fix positive constants γ and β satisfying $\max_S w(S) \leq \gamma \leq \beta$ and let $w_{ij} \leq 1$. If

$$\ell_{\text{clique}}(\mathcal{D};G) \triangleq \gamma - (\beta+1) \sum_{(v_i,v_j)\in E} w_{ij} p_i p_j + \frac{\beta}{2} \sum_{v_i \neq v_j} p_i p_j < (1-t)\beta,$$

then, with probability at least t, set $S^* \sim \mathcal{D}$ is a clique of weight $w(S^*) > \gamma - \ell_{\text{clique}}(\mathcal{D};G)/(1-t)$.

Proof. To ensure that the loss function is non-negative, we will work with the translated objective function $f(S;G) = \gamma - w(S)$, where the term γ is any upper bound of $w(S)$ for all S.

Theorem 1 guarantees that if

$$\mathbb{E}\left[f(S;G)\right] + P(S \notin \Omega)\,\beta = \ell_{\text{clique}}(\mathcal{D};G) < (1-t)\beta \tag{A.4}$$

and as long as $\max_S f(S;G) = \gamma - \min_S w(S) \leq \gamma \leq \beta$, then with probability at least t, set $S^* \sim \mathcal{D}$ satisfies $\gamma - \ell_{\text{clique}}(\mathcal{D};G)/(1-t) < w(S^*)$.

Denote by x_i a Bernoulli random variable with probability p_i. It is not difficult to see that

$$\mathbb{E}[w(S)] = \mathbb{E}\left[\sum_{(v_i,v_j)\in E} w_{ij} x_i x_j\right] = \sum_{(v_i,v_j)\in E} w_{ij} p_i p_j \tag{A.5}$$

We proceed to bound $P(S \notin \Omega_{\text{clique}})$. Without loss of generality, suppose that the edge weights have been normalized to lie in $[0,1]$. We define $\bar{w}(S)$ to be the weight of S on the complement graph:

$$\bar{w}(S) \triangleq \sum_{v_i,v_j\in S} \mathbf{1}_{\{(v_i,v_j)\notin E\}}$$

By definition, we have that $P\left(S \notin \Omega_{\text{clique}}\right) = P\left(\bar{w}(S) \geq 1\right)$. Markov's inequality then yields

$$P\left(S \notin \Omega_{\text{clique}}\right) \leq \mathbb{E}[\bar{w}(S)] = \mathbb{E}\left[\frac{|S|(|S|-1)}{2}\right] - \mathbb{E}[w(S)] \tag{A.6}$$

$$= \frac{1}{2}\mathbb{E}\left[\left(\sum_{v_i\in V} x_i\right)^2 - \sum_{v_i\in V} x_i\right] - \mathbb{E}[w(S)]$$

$$= \frac{1}{2}\sum_{v_i\neq v_j} \mathbb{E}\left[x_i x_j\right] + \frac{1}{2}\sum_{v_i\in V} \mathbb{E}\left[x_i^2\right] - \sum_{v_i\in V} \mathbb{E}[x_i] - \frac{1}{2}\mathbb{E}[w(S)]$$

$$= \frac{1}{2}\sum_{v_i\neq v_j} p_i p_j + \frac{1}{2}\sum_{v_i\in V} p_i - \frac{1}{2}\sum_{v_i\in V} p_i - \mathbb{E}[w(S)] = \frac{1}{2}\sum_{v_i\neq v_j} p_i p_j - \mathbb{E}[w(S)]. \tag{A.7}$$

It follows from the above derivations that

$$\gamma - \mathbb{E}[w(S)] + P(S \notin \Omega)\,\beta \leq \gamma - \mathbb{E}[w(S)] + \frac{\beta}{2}\sum_{v_i\neq v_j} p_i p_j - \beta\mathbb{E}[w(S)]$$

$$= \gamma - (1+\beta)\mathbb{E}[w(S)] + \frac{\beta}{2}\sum_{v_i\neq v_j} p_i p_j$$

$$= \gamma - (1+\beta)\sum_{(v_i,v_j)\in E} w_{ij} p_i p_j + \frac{\beta}{2}\sum_{v_i\neq v_j} p_i p_j. \tag{A.8}$$

The final expression is exactly the probabilistic loss function for the maximum clique problem.
□

A.1.3 Proof of Corollary 2

Corollary 2. Let the probabilities $p_1, \ldots, p_n$ giving rise to $\mathcal{D}$ be re-scaled such that $\sum_{v_i\in V} d_i p_i = \frac{v_l+v_h}{2}$ and, further, fix $\ell_{\text{cut}}(\mathcal{D};G) \triangleq \sum_{v_i\in V} d_i p_i - 2\sum_{(v_i,v_j)\in E} p_i p_j w_{ij}$. Set $S^* \sim \mathcal{D}$ satisfies

$$\text{cut}\left(S^*\right) < \ell_{\text{cut}}(\mathcal{D};G)/(1-t) \quad \text{and} \quad \text{vol}\left(S^*\right) \in [v_l, v_h],$$

with probability at least $t - 2\exp\left(-(v_h - v_l)^2/\sum_i 2d_i^2\right)$.

Proof. Denote by S the set of nodes belonging to the cut, defined as $S = \{v_i \in V$, such that $x_i = 1\}$. Our first step is to re-scale the probabilities such that, in expectation, the following is satisfied:

$$\mathbb{E}[\mathrm{vol}(S)] = \frac{v_l + v_h}{2}.$$

This can be achieved by noting that the expected volume is

$$\mathbb{E}[\mathrm{vol}(S)] = \mathbb{E}\left[\sum_{v_i \in V} d_i x_i\right] = \sum_{v_i \in V} d_i p_i$$

and then using the procedure described in Section A.3.

With the probabilities $p_1, \ldots, p_n$ re-scaled, we proceed to derive the probabilistic loss function corresponding to the min cut.

The cut of a set $S \sim \mathcal{D}$ can be expressed as

$$\mathrm{cut}(S) = \sum_{v_i \in S} \sum_{v_j \notin S} w_{ij} = \sum_{(v_i, v_j) \in E} w_{ij} z_{ij}, \tag{A.9}$$

where z_{ij} is a Bernoulli random variable with probability p_i which is equal to one if exactly one of the nodes v_i, v_j lies within set S. Formally,

$$z_{ij} = |x_i - x_j| = \begin{cases} 1 & \text{with probability } p_i - 2p_i p_j + p_j \\ 0 & \text{with probability } 2p_i p_j - (p_i + p_j) + 1 \end{cases} \tag{A.10}$$

It follows that the expected cut is given by

$$\begin{aligned} \mathbb{E}[\mathrm{cut}(S)] &= \sum_{(v_i, v_j) \in E} w_{ij} \mathbb{E}[z_{ij}] \\ &= \sum_{(v_i, v_j) \in E} w_{ij}(p_i - 2p_i p_j + p_j) \\ &= \sum_{(v_i, v_j) \in E} w_{ij}(p_i + p_j) - 2 \sum_{(v_i, v_j) \in E} p_i p_j w_{ij} = \sum_{v_i \in V} d_i p_i - 2 \sum_{(v_i, v_j) \in E} p_i p_j w_{ij}. \end{aligned} \tag{A.11}$$

We define, accordingly, the min-cut probabilistic loss as

$$\ell_{\mathrm{cut}}(\mathcal{D}; G) = \sum_{v_i \in V} d_i p_i - 2 \sum_{(v_i, v_j) \in E} p_i p_j w_{ij}$$

Then, for any $t \in (0, 1]$, Markov's inequality yields:

$$P\left(\mathrm{cut}(S) < \frac{\ell_{\mathrm{cut}}(\mathcal{D}; G)}{1 - t}\right) > t$$

The proof then concludes by invoking Theorem 2. □

A.1.4 Proof of theorem 3

Theorem 3. The minima of the probabilistic penalty loss are the minima of the constrained optimization problem:

$$\min_{\mathcal{D}} \ell(\mathcal{D}; G) = \min_{S\in\Omega} f(S; G).$$

Proof. First, observe that for $\mathcal{D} = \delta_S$, where δ_S is the point mass on any set $S \subseteq V$, we have $\mathbb{E}_{S\sim\mathcal{D}}[f(S)] = f(S)$. Therefore, for any $S \in \Omega$ and $\mathcal{D} = \delta_S$, the constraint term is zero and we see that

$$\ell(\mathcal{D}; G) = \mathbb{E}_{S\sim\mathcal{D}}[f(S)] = f(S; G), \tag{A.12}$$

and consequently

$$\min_{\mathcal{D}=\delta_{S\in\Omega}} \ell(\mathcal{D}; G) = \min_{S\in\Omega} f(S; G). \tag{A.13}$$

More generally, for any distribution $\mathcal{D}_{\text{feasible}}$ supported only on feasible solutions we have

$$\begin{aligned}
\ell(\mathcal{D}_{\text{feasible}}; G) &= \mathbb{E}[f(S; G)] \\
&\geq \sum_{S\in\Omega} P(S) \min_{S\in\Omega} f(S; G) \\
&= \min_{S\in\Omega} f(S; G).
\end{aligned}$$

In the general case, for any $\mathcal{D}$, we have

$$\begin{aligned}
\ell(\mathcal{D}; G) &= \sum_{S\subseteq V} P(S) f(S; G) + \beta P(S \notin \Omega) \\
&= \sum_{S\in\Omega} P(S) f(S; G) + \sum_{S\notin\Omega} P(S) f(S; G) + \beta P(S \notin \Omega) \\
&= \sum_{S\in\Omega} P(S) f(S; G) + \sum_{S\notin\Omega} P(S) f(S; G) + \beta \sum_{S\notin\Omega} P(S) \\
&\geq \sum_{S\notin\Omega} P(S)(f(S; G) + \beta) + \sum_{S\in\Omega} P(S) \min_{S\in\Omega} f(S; G) \\
&= \sum_{S\notin\Omega} P(S) \big(f(S; G) + \beta\big) + \min_{S\in\Omega} f(S; G)(1 - \sum_{S\notin\Omega} P(S)) \\
&= \underbrace{\sum_{S\notin\Omega} P(S) \left(f(S; G) + \beta - \min_{S\in\Omega} f(S; G)\right)}_{\alpha\geq 0} + \min_{S\in\Omega} f(S; G).
\end{aligned}$$

Thus, the probabilistic penalty function is bounded from below by the minimum of the constrained optimization problem. If the support of $\mathcal{D}$ contains only feasible solutions, then $P(S) = 0$ for $S \notin \Omega$. This implies $\ell(\mathcal{D}_{\text{feasible}}; G) \geq \min_{S\in\Omega} f(S)$. From A.13 we know that equality with the lower bound can be achieved for $\mathcal{D} = \delta_{S\in\Omega}$. When infeasible solutions are supported, then $\ell(\mathcal{D}; G) \geq \min_{S\in\Omega} f(S; G) + \alpha$ for $\alpha \in \mathbb{R}^+$. This establishes that the minima of the probabilistic penalty loss are the same as the minima of the constrained optimization problem. □

A.1.5 Proof of Corollary 4.

Let $y_1, y_2, \dots, y_m > 0$ be parameters corresponding to bad events $b_1, b_2, \dots, b_m$. Let $P(b_i)$ be the probability of a bad event b_i and let $\mathcal{N}_i^+$ be the neighborhood of b_i in the dependency graph, including b_i. If for all b_i, we have

$$P(S \in b_i) \leq \frac{y_i}{\prod_{j \in \mathcal{N}^+}(y_j + 1)},$$

Then $P(S \in \bigcap_{i \in \mathcal{C}} \bar{b_i}) > 0$.

Proof. The proof can be obtained from the conditions of the asymmetric LLL.

$$\begin{aligned} P(S \in b_i) &\leq \lambda_i \prod_{j \in \mathcal{N}_i} (1 - \lambda_j), \\ &= \frac{\lambda_i}{\prod_{j \in \mathcal{N}_i} \left(\frac{1}{(1-\lambda_j)}\right)} \\ &= \frac{\lambda_i}{(1-\lambda_i)\prod_{j \in \mathcal{N}_i^+} \left(\frac{1}{(1-\lambda_j)}\right)} \\ &= \frac{\lambda_i}{(1-\lambda_i)\prod_{j \in \mathcal{N}_i^+} \left(1 + \frac{\lambda_j}{(1-\lambda_j)}\right)}, \end{aligned}$$

By setting $y_i = \frac{\lambda_i}{1-\lambda_i}$ for all i we obtain Dobrushin's condition. Therefore, it follows from the asymmetric LLL that if Dobrushin's condition holds, then $P(S \in \bigcap_{i \in \mathcal{C}} \bar{b_i}) > 0$. □

A.2 Experimental details

A.2.1 Datasets

The following table presents key statistics of the datasets that were used in this study:

To speed up computation and training, for the Facebook dataset, we kept graphs consisting of at most 15000 nodes (i.e., 70 out of the total 100 available graphs of the dataset).

The RB test set can be downloaded from the following link: https://www.dropbox.com/s/9bdq1y69dw1q77q/cliques_test_set_solved.p?dl=0. The latter was generated using the procedure described by Xu (2007). We used a python implementation by Toenshoff et al. (2019) that is available on the RUN-CSP repository: https://github.com/RUNCSP/RUN-CSP/blob/master/generate_xu_instances.py. Since the parameters of the original training set were not

	IMDB	COLLAB	TWITTER	RB (Train)	RB (Test)	RB (Large Inst.)	SF-295	FACEBOOK
nodes	19.77	74.49	131.76	216.673	217.44	1013.25	26.06	7252.71
edges	96.53	2457.78	1709.33	22852	22828	509988.2	28.08	276411.19
reduction time	0.0003	0.006	0.024	0.018	0.018	0.252	–	–
number of test graphs	200	1000	196	2000	500	40	8055	14

Table A.1: Average number of nodes and edges for the considered datasets. Reduction time corresponds to the average number of seconds needed to reduce a maximum clique instance to a maximum independent instance. Number of test graphs refers to the number of graphs that the methods were evaluated on, in a given dataset.

available, we selected a set of initial parameters such that the generated dataset resembles the original training set. As seen in Table A.1, the properties of the generated test set are close to those of the training set. Specifically, the training set contained graphs whose size varied between 50 and 500 nodes and featured cliques of size 5 to 25. The test set was made out of graphs whose size was between 50 and 475 nodes and contained cliques of size 10 to 25. These minor differences provide a possible explanation for the drop in test performance of all methods (larger cliques tend to be harder to find).

All other datasets are publicly available.

A.2.2 Neural network architecture

In both problems, Erdős' GNN and our own neural baselines were given as node features a one-hot encoding of a random node from the input graph. For the local graph partitioning setting, our networks consisted of 6 GIN layers followed by a multi-head GAT layer. The depth was kept constant across all datasets. We employed skip connections and batch-normalization at every layer. For the maximum clique problem, we also incorporated graph size normalization for each convolution, as we found that it improved optimization stability. The networks in this setting did not use a GAT layer, as we found that multi-head GAT had a negative impact on the speed/memory of the network, while providing only negligible benefits in accuracy. Furthermore, locality was enforced after each layer by masking the receptive field. That is, after 1 layer of convolution only 1-hop neighbors were allowed to have nonzero values, after 2 layers only 2-hop neighbors could have nonzero values, etc. The output of the final GNN layer was passed through a two layer perceptron giving as output one value per node. The aforementioned numbers were re-scaled to lie in $[0,1]$ (using a graph-wide min-max normalization) and were interpreted as probabilities $p_1,\dots,p_n$. In the case of local graph partitioning, the forward-pass was concluded by the appropriate re-scaling of the probabilities (as described in Section 2.2.2).

A.2.3 Local graph partitioning setup

Following the convention of local graph clustering algorithms, for each graph in the test set we randomly selected d nodes of the input graph to act as cluster *seeds*, where $d = 10, 30$, and 100 for SF-295, TWITTER, and FACEBOOK, respectively. Each method was run once for each seed resulting in d sets per graph. We obtained one number per seed by averaging the conductances of the graphs. Table 2.3 reports the mean and standard deviation of these numbers. The correct procedure is the one described here.

The volume-constrained graph partitioning formulation can be used to minimize conductance as follows: Perform grid search over the range of feasible volumes and create a small interval around each target volume. Then, solve a volume-constrained partitioning problem for each interval, and return the set of smallest conductance identified.

We used a fast and randomized variant of the above procedure with all neural approaches and Gurobi (see Section 2.4.7 for more details). Specifically, for each seed node we generated a random volume interval within the receptive field of the network, and solved the corresponding constrained partitioning problem. Our construction ensured that the returned sets always contained the seed node and had a controlled volume. For L1 and L2 GNN, we obtained the set by sampling from the output distribution. We drew 10 samples and kept the best. We found that in contrast to flat thresholding (like in the maximum clique), sampling yielded better results in this case.

For the parameter search of local graph clustering methods, we found the best performing parameters on a validation set via grid search when that was appropriate. For CRD, we searched for all the integer values in the [1,20] interval for all 3 of the main parameters of the algorithm. For Simple Local, we searched in the [0,1] interval for the locality parameter. Finally, for Pagerank-Nibble we set a lower bound on the volume that is 10 % of the total graph volume. It should be noted, that while local graph clustering methods achieved inferior conductance results, they do not require explicit specification of a receptive field which renders them more flexible.

A.2.4 Hardware and software

All methods were run on an Intel Xeon Silver 4114 CPU, with 192GB of available RAM. The neural networks were executed on a single RTX TITAN 25GB graphics card. The code was executed on version 1.1.0 of PyTorch and version 1.2.0 of PyTorch Geometric.

A.3 Iterative scheme for non-linear re-scaling

Denote by $\mathcal{D}^0$ the distribution of sets predicted by the neural network and let $p_1^0, \ldots, p_n^0$ be the probabilities that parameterize it. We aim to re-scale these probabilities such that the constraint is satisfied in expectation:

$$\sum_{v_i \in V} a_i p_i = \frac{b_l + b_h}{2}, \quad \text{where} \quad p_i = \text{clamp}\big(c\, p_i^0, 0, 1\big) \quad \text{and} \quad c \in \mathbb{R}.$$

This can be achieved by iteratively applying the following recursion:

$$p_i^{\tau+1} \leftarrow \text{clamp}\big(c^\tau p_i^\tau, 0, 1\big), \quad \text{with} \quad c^\tau = \frac{b - \sum_{v_i \in Q^\tau} a_i}{\sum_{v_i \in V \setminus Q^\tau} a_i p_i^\tau} \quad \text{and} \quad Q^\tau = \{v_i \in V : p_i^\tau = 1\},$$

where $b = \frac{b_l + b_h}{2}$.

The fact that convergence occurs can be easily deduced. Specifically, consider any iteration τ and let Q^τ be as above. If $p_i^{\tau+1} < 1$ for all $v_i \in V \setminus Q^\tau$, then the iteration has converged. Otherwise, we will have $Q^\tau \subset Q^{\tau+1}$. From the latter, it follows that in every τ (but the last), set Q^τ must expand until either $\text{clamp}\big(c^\tau p_i^\tau, 0, 1\big) = b$ or $Q^\tau = V$. The latter scenario will occur if $\sum_{v_i \in V} a_i \leq b$.

B Appendix of Chapter 3

B.1 Examples of extensions, proofs, and experimental details

B.1.1 Lovász extension.

Recall the definition: $\mathbf{x}$ is sorted so that $x_1 \geq x_2 \geq \ldots \geq x_d$. Then the Lovász extension corresponds to taking $S_i = \{1, \ldots, i\}$, and letting $p_{\mathbf{x}}(S_i) = x_i - x_{i+1}$, the non-negative increments of $\mathbf{x}$ (where recall we take $x_{n+1} = 0$). All other sets have zero probability. For convenience, we introduce the shorthand notation $a_i = p_{\mathbf{x}}(S_i) = x_i - x_{i+1}$

Feasibility. Clearly all $a_i = x_i - x_{i+1} \geq 0$, and $\sum_{i=1}^n a_i = \sum_{i=1}^n (x_i - x_{i+1}) = x_1 \leq 1$. Any remaining probability mass is assigned to the empty set: $p_{\mathbf{x}}(\varnothing) = 1 - x_1$, which contributes nothing to the extension $\mathfrak{F}$ since $f(\varnothing) = 0$ by assumption. All that remains is to check that

$$\sum_{i=1}^n p_{\mathbf{x}}(S_i) \cdot \mathbf{1}_{S_i} = \mathbf{x}.$$

For a given $k \in [n]$, note that the only sets S_i with non-zero kth coordinate are $S_1, \ldots, S_k$, and in all cases $(\mathbf{1}_{S_i})_k = 1$. So the kth coordinate is precisely $\sum_{i=1}^k p_{\mathbf{x}}(S_i) = \sum_{i=1}^k (x_i - x_{i+1}) = x_k$, yielding the desired formula.

Extension. Consider an arbitrary $S \subseteq [n]$. Since we assume $\mathbf{x} = \mathbf{1}_S$ is sorted, it has the form $\mathbf{1}_S = (\underbrace{1, 1, \ldots, 1}_{k \text{ times}}, 0, 0, \ldots 0)^\top$. Therefore, for each $j < k$ we have $a_j = x_j - x_{j+1} = 1 - 1 = 0$ and for each $j > k$ we have $a_j = x_j - x_{j+1} = 0 - 0 = 0$. The only non-zero probability is $a_k = x_k - x_{k+1} = 1 - 0 = 1$. So,

$$\mathfrak{F}(\mathbf{1}_S) = \sum_{i=1}^n a_i f(S_i) = \sum_{i: i \neq k} a_i f(S_i) + a_k f(S_k) = 0 + 1 \cdot f(S_k) = f(S)$$

where the the final equality follows since by definition S_k corresponds exactly to the vector $(\underbrace{1, 1, \ldots, 1}_{k \text{ times}}, 0, 0, \ldots 0)^\top = \mathbf{1}_S$ and so $S_k = S$.

Continuity. The Lovász extension is well-known and its properties have been carefully studied. In particular, it is known to be a Lipschitz function (Bach, 2019). For completeness, we provide a simple proof here.

Lemma 3. Let $p_{\mathbf{x}}$ be as defined for the Lovász extension. Then $\mathbf{x} \mapsto p_{\mathbf{x}}(S)$ is Lipschitz for all

$S \subseteq [n]$.

Proof. First note that $p_{\mathbf{x}}$ is piecewise linear, with one piece per possible ordering $x_1 \geq x_2 \geq \ldots \geq x_n$ (so $n!$ pieces in total). Within the interior of each piece $p_{\mathbf{x}}$ is linear, and therefore Lipschitz. So in order to prove global Lipschitzness, it suffices to show that $p_{\mathbf{x}}$ is continuous at the boundaries between pieces (the Lipschitz constant is then the maximum of the Lipschitz constants for each linear piece).

Now consider a point $\mathbf{x}$ with $x_1 \geq \ldots \geq x_i = x_{i+1} \geq \ldots \geq x_n$. Consider the perturbed point $\mathbf{x}_\delta = \mathbf{x} - \delta \mathbf{e}_i$ with $\delta > 0$, and $\mathbf{e}_i$ denoting the ith standard basis vector. To prove continuity of $p_{\mathbf{x}}$ it suffices to show that for any S we have $p_{\mathbf{x}_\delta}(S) \to p_{\mathbf{x}}(S)$ as $\delta \to 0^+$.

There are two sets in the support of $p_{\mathbf{x}}$ whose probabilities are different under $p_{\mathbf{x}_\delta}$, namely: $S_i = \{1, \ldots, i\}$ and $S_{i+1} = \{1, \ldots, i, i+1\}$. Similarly, there are two sets in the support of $p_{\mathbf{x}_\delta}$ whose probabilities are different under $p_{\mathbf{x}}$, namely: $S'_i = \{1, \ldots, i-1, i+1\}$ and $S'_{i+1} = \{1, \ldots, i, i+1\} = S_{i+1}$. So it suffices to show the convergence $p_{\mathbf{x}_\delta}(S) \to p_{\mathbf{x}}(S)$ for these four S. Consider first S_i:

$$\left|p_{\mathbf{x}_\delta}(S_i) - p_{\mathbf{x}}(S_i)\right| = \left|0 - (x_i - x_{i+1})\right| = 0$$

where the final equality uses the fact that $x_i = x_{i+1}$. Next consider $S_{i+1} = S'_{i+1}$:

$$\left|p_{\mathbf{x}_\delta}(S_{i+1}) - p_{\mathbf{x}}(S_{i+1})\right| = \left|(x'_{i+1} - x'_{i+2}) - (x_{i+1} - x_{i+2})\right| = \left|(x'_{i+1} - x_{i+1}) - (x'_{i+2} - x_{i+2})\right| = 0$$

Finally, we consider S'_i:

$$\begin{aligned}\left|p_{\mathbf{x}_\delta}(S'_i) - p_{\mathbf{x}}(S'_i)\right| &= \left|(x'_i - x'_{i+1}) - (x_i - x_{i+1})\right| \\ &= \left|(x'_{i+1} - x_{i+1}) - (x'_{i+1} - x_{i+1})\right| \\ &= \left|(x_{i+1} - \delta - x_{i+1}) - (x'_{i+1} - x_{i+1})\right| \\ &= \delta \to 0\end{aligned}$$

completing the proof. □

B.1.2 Bounded cardinality Lovaśz extension.

The bounded cardinality extension considers n sets S of cardinality at most k, with $n \geq k \geq 2$. We collect $\{S_i\}_{i=1}^n$ of subsets of $[n]$ in an $n \times n$ matrix $\mathbf{S} \in \{0,1\}^{n\times n}$ whose ith column is $\mathbf{1}_{S_i}$:

$$\mathbf{S} = \begin{bmatrix} \overbrace{1 & \dots & 1}^{k} & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \ddots & \ddots & 1 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The matrix will contain k sets of gradually increasing cardinality, from 1 up until k, and $n-k$ sets of cardinality exactly k. In this notation, the dual LP constraint $\sum_{S\subseteq[n]} y_S \mathbf{1}_S = \mathbf{x}$ can be written as $\mathbf{Sp} = \mathbf{x}$, where the ith coordinate of $\mathbf{p}$ defines $p_{\mathbf{x}}(S_i)$. Then, the bounded cardinality extension coefficients $p_{\mathbf{x}}(S)$ are the coordinates of the vector $\mathbf{y}$, where $\mathbf{y} = \mathbf{S}^{-1}\mathbf{x}$. To calculate the inverse, we will leverage the fact that $\mathbf{S}$ will be triangular Toeplitz by construction. Clearly, its inverse will also be triangular.

Lemma 4. The entries (i,j) of the inverse are

$$\mathbf{S}^{-1}(i,j) = \begin{cases} 1, \text{ if } (j-i) \bmod k = 0 \text{ and } i \leq j, \\ -1, \text{ if } (j-i) \bmod k = 1 \text{ and } i \leq j, \\ 0, \text{ otherwise,} \end{cases} \tag{B.1}$$

for $i = 1,2,\dots,n$.

Proof. The proof relies on known results for banded Toeplitz matrices. A banded Toeplitz matrix of bandwidth r and superdiagonal s is an $n \times n$ matrix that has the following form:

$$\mathbf{T}_{r,s} = \begin{bmatrix} c_{s+1} & c_s & \dots & c_1 & & 0 \\ c_{s+2} & c_{s+1} & c_s & \dots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & & c_1 \\ c_r & & \ddots & \ddots & & \vdots \\ & \ddots & & \ddots & \ddots & \vdots \\ 0 & & c_r & \dots & \dots & c_{s+1} \end{bmatrix}.$$

Note here that the (i,j) entry of $\mathbf{T}$, due to its Toeplitz structure, is going to be $\mathbf{T}(i,j) = c_{i-j+s+1}$. For convenience, we are going to invert $\mathbf{S}^\top$ and the result straightforwardly transfers to $\mathbf{S}$. For $\mathbf{S}^\top$, we have superdiagonal $s=0$ and bandwidth $r=k$. It is known (Meek, 1983; Trench, 1974) that the entries $g_{i-j+1} = (\mathbf{S}^\top)^{-1}(i,j)$ of the inverse will obey the following difference equation:

$$c_k g_{l-k} + c_{k-1} g_{l-k+1} + \cdots = 0, \quad l \geq 3, \quad g_1 = 1,$$

with $g_0 = g_{-1} = \cdots = g_{3-k} = 0$. Considering the conditions above and the fact that $c_1 = c_2 = \cdots = c_k = 1$, the difference equation simplifies to

$$\sum_{t=0}^{k-1} g_{l-k+t} = 0. \tag{B.2}$$

As an example, let us compute the case for $k = 3, l = 3$. We obtain $g_0 + g_1 + g_2 = 0$, which implies $g_2 = -1$. It is easy to see that for any k, computing the difference equation for $l = 3$ yields $g_2 = -1$ since all the negative indices do not contribute to the sum, reducing it to $g_1 + g_2 = 0$.

We continue with $k = 3, l = 4$ and obtain $g_1 + g_2 + g_3 = 0$, which implies $g_3 = 0$. By incrementing l, observe that we are shifting the terms in the sum by one, so this straightforwardly implies that $l = 5$ yields $g_4 = 1$ for $k = 3$, and so on. Generalizing this observation, we obtain the following cases:

- $g_t = 1$, for $t = mk + 1$,
- $g_t = -1$, for $t = mk + 2$,
- $g_t = 0$, otherwise.

Here, m is a non-negative integer. The lemma follows straightforwardly from that observation.

$\square$

Equivalence to the Lovaśz extension. We want to show that the bounded cardinality extension is equivalent to the Lovaśz extension when $k = n$. Let $T_{i,k} = \{j \mid (j-i) \bmod k = 0, \text{ for } i \leq j \leq n,\}$, i.e., $T_{i,k}$ stores the indices where $j - i$ is perfectly divided by k. From the analytic form of the inverse, observe that the i-th coordinate of $\mathbf{y}$ is $p_{\mathbf{x}}(S_i) = \sum_{j \in T_{i,k}} (x_j - x_{j+1})$. For $k = n$, we have $T_{i,n} = \{j \mid (j-i) \bmod n = 0\} = \{i\}$, and therefore $p_{\mathbf{x}}(S_i) = x_i - x_{i+1}$, which are the coefficients of the Lovász extension.

Feasibility. The equation $\mathbf{y} = \mathbf{S}^{-1}\mathbf{x}$ guarantees that the constraint $\mathbf{x} = \sum_{i=1}^n y_{S_i} \mathbf{1}_{S_i}$ is obeyed. Recall that $\mathbf{x}$ is sorted in descending order like in the case of the Lovász extension. Then, it is easy to see that $p_{\mathbf{x}}(S_i) = \sum_{j \in T_{i,k}} (x_j - x_{j+1}) \leq x_i$, because $x_i - x_{i+1}$ is always contained in the summation for $p_{\mathbf{x}}(S_i)$. We also have $p_{\mathbf{x}}(S_i) \geq 0$ because $x_i - x_{i+1} \geq 0$. Therefore, by restricting $\mathbf{x}$ in the probability simplex we could ensure $\sum_{i=1}^n p_{\mathbf{x}}(S_i) \leq \sum_{i=1}^n x_i = 1$. To secure tight equality, we allocate the rest of the mass to the empty set, i.e., $p_{\mathbf{x}}(\varnothing) = 1 - \sum_{i=1}^n p_{\mathbf{x}}(S_i)$, which does not affect the value of the extension since the corresponding Boolean is the zero vector. However, $\mathbf{x}$ in the simplex implies that we cannot obtain binary vectors of cardinality larger than 1 which is necessary for the extension property to hold for sets of size up to k. The following proposition provides a sufficient condition for the feasibility and extension property.

Proposition 9. Let $\mathbf{x} \in [0,1]$ with $\sum_{i=k+1}^n x_i \le 1 - x_1$. Then $p_{\mathbf{x}}(S_i) = \sum_{j \in T_{i,k}} (x_j - x_{j+1})$ defines a feasible solution to the scalar extension dual LP.

Proof. To guarantee feasibility, we need that $\sum_{i=1}^n p_{\mathbf{x}}(S_i) \le 1$. It is easy to see that $\sum_{i=1}^k p_{\mathbf{x}}(S_i) = x_1$ because the first k coefficients of the extension are identical to those of the Lovasz extension. This means that if $\sum_{i=k+1}^n p_{\mathbf{x}}(S_i) \le 1 - x_1$, then feasibility is guaranteed. Since $p_{\mathbf{x}}(S_i) = \sum_{j \in T_{i,k}} (x_j - x_{j+1})$, each coefficient $p_{\mathbf{x}}(S_i)$ depends on differences $x_j - x_{j+1}$ for $j \ge i$. Hence, we have that $\sum_{i=k+1}^n p_{\mathbf{x}}(S_i) \le \sum_{i=k+1}^n x_i$. Therefore, $\sum_{i=k+1}^n x_i \le 1 - x_1$ implies $\sum_{i=k+1}^n p_{\mathbf{x}}(S_i) \le 1 - x_1$. This guarantees that $\sum_{i=1}^n p_{\mathbf{x}}(S_i) \le 1$ and concludes the proof. $\square$

Extension. To prove the extension property we need to show that $\mathfrak{F}(\mathbf{1}_S) = f(S)$ for all S with $|S| \le k$. Consider any such set S and recall that we have sorted $\mathbf{1}_S$ with arbitrary tie breaks, such that $x_i = 1$ for $i \le |S|$ and $x_i = 0$ otherwise. Due to the equivalence with the Lovaśz extension, the extension property is guaranteed when $k = n$ for all possible sets. For $k < n$, consider the following three cases for $T_{i,k}$.

- When $i > |S|$, $T_{i,k} = \emptyset$ because for sorted $\mathbf{x}$ of cardinality at most k, we know for the coordinates that $x_i = x_{i+1} = 0$. For $i > k$, this implies that $p_{\mathbf{x}}(S_i) = 0$.
- When $i < |S|$, $\sum_{j \in T_{i,k}} (x_j - x_{j+1}) = 0$ because $x_j = x_{j+1} = 1$ and we have again $p_{\mathbf{x}}(S_i) = 0$.
- When $i = |S|$, observe that $\sum_{j \in T_{i,k}} (x_j - x_{j+1}) = x_i - x_{i+1} = x_i$. Therefore, $p_{\mathbf{x}}(S_i) = 1$. in that case.

Bringing it all together, $\mathfrak{F}(\mathbf{1}_S) = \sum_{i=1}^n p_{\mathbf{x}} f(S_i) = p_{\mathbf{x}}(S) f(S) = f(S)$ since the sum contains only one nonzero term, the one that corresponds to $i = |S|$.

Continuity. Similar to the Lovaśz extension, $p_{\mathbf{x}}$ in the bounded cardinality extension is piecewise linear and therefore a.e. differentiable with respect to $\mathbf{x}$, where each piece corresponds to an ordering of the coordinates of $\mathbf{x}$. On the other hand, unlike the Lovaśz extension, the mapping $\mathbf{x} \mapsto p_{\mathbf{x}}(S)$ is not necessarily globally Lipschitz when $k < n$, because it is not guaranteed to be Lipschitz continuous at the boundaries.

B.1.3 Singleton extension.

Feasibility. The singleton extension is not dual LP feasible. However, one of the key reasons why feasibility is important is that it implies Proposition 1, which show that optimizing $\mathfrak{F}$ is a reasonable surrogate to f. In the case of the singleton extension, however, Proposition 1 still holds even without feasibility for f. This includes the case of the training accuracy loss, which can be viewed as minimizing the set function $f(\{\hat{y}\}) = -\mathbf{1}\{y_i = \hat{y}\}$.

Here we give an alternative proof of Proposition 1 for the singleton extension. Consider the same assumptions as Proposition 1 with the additional requirement that $\min_S f(S) < 0$ (this merely asserts hat $S = \varnothing$ is not a trivial solution to the minimization problem, and that the minimizer of f is unique. This is true, for example, for the training accuracy objective we consider in Section 3.5.

Proof of Proposition 1 for singleton extension. For $\mathbf{x} \in \mathcal{X} = [0,1]^n$,

$$\begin{aligned}
\mathfrak{F}(\mathbf{x}) &= \sum_{i=1}^{n} p_{\mathbf{x}}(S_i) f(S_i) \\
&= \sum_{i=1}^{n} (x_i - x_{i+1}) f(S_i) \\
&\geq \sum_{i=1}^{n} (x_i - x_{i+1}) \min_{j \in [n]} f(S_j) \\
&\geq (x_1 - x_{n+1}) \min_{j \in [n]} f(S_j) \\
&\geq x_1 \cdot \min_{j \in [n]} f(S_j) \\
&\geq \min_{j \in [n]} f(S_j)
\end{aligned}$$

where the final inequality follows since $\min_{j\in[n]} f(S_j) < 0$. Taking $\mathbf{x} = (1,0,0,\ldots,0)^\top$ shows that all the inequalities can be made tight, and the first statement of Proposition 1 holds. For the second statement, suppose that $\mathbf{x} \in \mathcal{X} = [0,1]^n$ minimizes $\mathfrak{F}$. Then all the inequality in the preceding argument must be tight. In particular, tightness of the final inequality implies that $x_1 = 1$. Meanwhile, tightness of the first inequaliity implies that $x_i - x_{i+1} = 0$ for all i for which $f(S_i) \neq \min_{j\in[n]} f(S_j)$, and tightness of the second inequality implies that $x_{n+1} = 0$. These together imply that $\mathbf{x} = \mathbf{1} \oplus \mathbf{0}_{n-1}$ where $\mathbf{1}$ is a 1×1 vector with entry equal to one, and $\mathbf{0}_{n-1}$ is an all zeros vectors of length $n-1$, and $\oplus$ denotes concatenation. Since $f(S_1) = \min_{j\in[n]} f(S_j)$ is the unique minimizer we have that $\mathbf{x} = \mathbf{1}_{S_1} \in \mathrm{Hull}\big(\mathrm{argmin}_{\mathbf{1}_{S_i}: i \in [n]} f(S_i)\big)$, completing the proof. □

Extension. Consider an arbitrary $i \in [n]$. Since we assume $\mathbf{x} = \mathbf{1}_{\{i\}}$ is sorted, we are without loss of generality considering $\mathbf{1}_{\{1\}} = (1,0,\ldots,0,0,\ldots 0)^\top$. Therefore, we have $p_{\mathbf{x}}(S_1) = x_1 - x_2 = 1 - 0 = 1$ and for each $j > 1$ we have $p_{\mathbf{x}}(S_j) = x_j - x_{j+1} = 0 - 0 = 0$. The only non-zero probability is $p_{\mathbf{x}}(S_1)$, and so

$$\mathfrak{F}(\mathbf{1}_{\{1\}}) = \sum_{j=1}^{n} p_{\mathbf{x}}(S_j) f(S_j) = f(S_1) = f(\{1\}).$$

Continuity. The proof of continuity of the singleton extension is a simple adaptation of the proof used for the Lovaśz extension, which we omit.

B.1.4 Permutations and Involutory Extension.

Feasibility. It is known that every elementary permutation matrix is involutory, i.e., $\mathbf{SS} = \mathbf{I}$. Given such an elementary permutation matrix $\mathbf{S}$, since $\mathbf{S}(\mathbf{Sx}) = \mathbf{S}p_{\mathbf{x}} = \mathbf{x}$, the constraint $\sum_{S\subseteq[n]} y_S \mathbf{1}_S = \mathbf{x}$ is satisfied. Furthermore, $\sum_{S\subseteq[n]} y_S = 1$ can be secured if $\mathbf{x}$ is in the simplex, since the sum of the elements of a vector is invariant to permutations of the entries.

Extension. If the permutation has a fixed point at the maximum element of $\mathbf{x}$, i.e., it maps the maximum element to itself, then any elementary permutation matrix with such a fixed point yields an extension on singleton vectors. Without loss of generality, let $\mathbf{x} = \mathbf{e}_1$, where $\mathbf{e}_1$ is the standard basis vector in $\mathbb{R}^n$. Then $\mathbf{Se}_1 = \mathbf{e}_1$ and therefore $p_{\mathbf{x}}(\mathbf{e}_1) = 1$. This in turn implies $\mathfrak{F}(\mathbf{e}_1) = 1 \cdot f(\mathbf{e}_1)$. This argument can be easily applied to all singleton vectors.

Continuity. The permutation matrix $\mathbf{S}$ can be chosen in advance for each $\mathbf{x}$ in the simplex. Since $p_{\mathbf{x}} = \mathbf{Sx}$, the probabilities are piecewise-linear and each piece is determined by the fixed point induced by the maximum element of $\mathbf{x}$. Consequently, $p_{\mathbf{x}}$ depends continuously on $\mathbf{x}$.

B.1.5 Multilinear extension.

Recall that the multiliniear extension is defined via $p_{\mathbf{x}}(S) = \prod_{i\in S} x_i \prod_{i\notin S}(1-x_i)$ supported on all subsets $S \subseteq [n]$ in general.

Feasibility. The definition of $p_{\mathbf{x}}(S)$ is equivalent to:

$$p_{\mathbf{x}}(S) = \prod_{i=1}^{n} x_i^{y_i}(1-x_i)^{1-y_i}$$

where $y_i = 1$ if $i \in S$ and zero otherwise. That is, $p_{\mathbf{x}}(S)$ is the product of n independent Bernoulli distributions. So we clearly have $p_{\mathbf{x}}(S) \geq 0$ and $\sum_{S\subseteq[n]} p_{\mathbf{x}}(S) = 1$. The final feasibility condition, that $\sum_{S\subseteq[n]} p_{\mathbf{x}}(S)\cdot \mathbf{1}_S = \mathbf{x}$ can be checked by induction on n. For $n = 1$ there are only two sets: $\{1\}$ and the empty set. And clearly $p_{\mathbf{x}}(\{1\})\cdot \mathbf{1}_{\{1\}} = x_1(1-x_1)^0 = x_1$, so we have the base case.

Extension. For any $S \subseteq [n]$ we have $p_{\mathbf{1}_S}(S) = \prod_{i\in S} x_i \prod_{i\notin S}(1-x_i) = \prod_{i\in S} 1 \prod_{i\notin S}(1-0) = 1$. So $\mathfrak{F}(\mathbf{1}_S) = \mathbb{E}_{T\sim p_{\mathbf{x}}} f(T) = f(S)$.

Continuity. Fix and $S \subseteq [n]$. Again we check Lipschitzness. We use ∂_{x_k} to denote the derivative operator with respect to x_k. If $k \in S$ we have

$$\left|\partial_{x_k} p_{\mathbf{1}_S}(S)\right| = \left|\partial_{x_k} \prod_{i\in S} x_i \prod_{i\notin S}(1-x_i)\right| = \prod_{i\in S\setminus\{k\}} x_i \prod_{i\notin S}(1-x_i) \leq 1.$$

Similarly, if $k \notin S$ we have,

$$\left|\partial_{x_k} p_{\mathbf{1}_S}(S)\right| = \left|\partial_{x_k} \prod_{i\in S} x_i \prod_{i\notin S}(1-x_i)\right| = \left|-\prod_{i\in S} x_i \prod_{i\notin S\cup\{k\}}(1-x_i)\right| \leq 1.$$

Hence the spectral norm of the Jacobian $Jp_{\mathbf{x}}(S)$ is bounded, and so $\mathbf{x} \mapsto p_{\mathbf{x}}(S)$ is a Lipschitz map.

B.2 Neural Set Function Extensions

B.2.1 Containment and Extension properties

This section re-states and proves the results from Section 3.4. To start, recall the definition of the primal LP:

$$\max_{\mathbf{z},b}\{\mathbf{x}^\top \mathbf{z} + b\}, \text{ where } (\mathbf{z}, b) \in \mathbb{R}^n \times \mathbb{R} \text{ and } \mathbf{1}_S^\top \mathbf{z} + b \leq f(S) \text{ for all } S \subseteq [n].$$

and primal SDP:

$$\max_{\mathbf{Z}\succeq 0, b\in\mathbb{R}} \{\mathrm{Tr}(\mathbf{X}^\top \mathbf{Z}) + b\} \text{ subject to } \frac{1}{2}\mathrm{Tr}((\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top)\mathbf{Z}) + b \leq f(S\cap T) \text{ for } S, T \subseteq [n]. \tag{B.3}$$

Proposition 10. (Mapping LP feasible solutions to SDP feasible solutions) For any $\mathbf{x} \in [0,1]^n$, define $\mathbf{X} = \sqrt{\mathbf{x}}\sqrt{\mathbf{x}}^\top$ with the square-root taken entry-wise. Then, for any $(\mathbf{z}, b) \in \mathbb{R}^n_+ \times \mathbb{R}$ that is primal LP feasible, the pair $(\mathbf{Z}, b)$ where $\mathbf{Z} = \mathrm{diag}(\mathbf{z})$, is primal SDP feasible and the objective values agree: $\mathrm{Tr}(\mathbf{X}^\top \mathbf{Z}) = \mathbf{z}^\top \mathbf{x}$.

Proof. We start with the feasibility claim. Suppose that $(\mathbf{z}, b) \in \mathbb{R}^n_+ \times \mathbb{R}$ is a feasible solution to the primal LP. We must show that $(\mathbf{Z}, b)$ is a feasible solution to the primal SDP with $\mathbf{X} = \sqrt{\mathbf{x}}\sqrt{\mathbf{x}}^\top$ and where $\mathbf{Z} = \mathrm{diag}(\mathbf{z})$.

Recall the general formula for the trace of a matrix product: $\mathrm{Tr}(\mathbf{AB}) = \sum_{i,j} A_{ij}B_{ji}$. With this in mind, and noting that the (i, j) entry of $\mathbf{1}_S\mathbf{1}_T^\top$ is equal to 1 if $i, j \in S \cap T$, and zero otherwise, we have for any $S, T \subseteq [n]$ that

$$\begin{aligned}\frac{1}{2}\mathrm{Tr}((\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top)\mathbf{Z}) + b = \mathrm{Tr}(\mathbf{1}_S\mathbf{1}_T^\top\mathbf{Z}) + b &= \sum_{i,j=1}^{n} (\mathbf{1}_S\mathbf{1}_T^\top)_{ij} \cdot \mathrm{diag}(\mathbf{z})_{ij} + b \\ &= \sum_{i,j\in S\cap T} (\mathbf{1}_S\mathbf{1}_T^\top)_{ij} \cdot \mathrm{diag}(\mathbf{z})_{ij} + b \\ &= \sum_{i,j\in S\cap T} \mathrm{diag}(\mathbf{z})_{ij} + b \\ &= \sum_{i\in S\cap T} z_i + b \\ &= \mathbf{1}_{S\cap T}^\top\mathbf{z} + b \\ &\le f(S\cap T)\end{aligned}$$

showing SDP feasibility. That the objective values agree is easily seen since:

$$\mathrm{Tr}(\mathbf{ZX}) = \sum_{i,j=1}^{n} \mathrm{diag}(\mathbf{z})_{ij} \cdot \sqrt{x_i}\sqrt{x_j} = \sum_{i=1}^{n} z_i \cdot \sqrt{x_i}\sqrt{x_i} = \mathbf{x}^\top\mathbf{z}.$$

□

Next, we provide a proof for the construction of neural extensions. Recall the statement of the main result.

Proposition 11. Let $p_{\mathbf{x}}$ induce a scalar SFE of f. For $\mathbf{X} \in \mathbb{S}_+^n$ with distinct eigenvalues, consider the decomposition $\mathbf{X} = \sum_{i=1}^n \lambda_i \mathbf{x}_i\mathbf{x}_i^\top$ and fix

$$p_{\mathbf{X}}(S,T) = \sum_{i=1}^{n} \lambda_i\, p_{\mathbf{x}_i}(S) p_{\mathbf{x}_i}(T) \text{ for all } S, T \subseteq [n]. \tag{B.4}$$

Then, $p_{\mathbf{X}}$ defines a neural SFE $\mathfrak{F}$ at $\mathbf{X}$.

Proof. We begin by showing through the eigendecomposition of $\mathbf{X}$ that the $\mathfrak{F}$ defined by $p_{\mathbf{X}}(S,T)$ is dual SDP feasible. It is clear that $\sum_{S,T} p_{\mathbf{X}}(S,T) = 1$ as long as $\sum_{i=1}^n \lambda_i = 1$, which can be easily enforced by appropriate normalization of $\mathbf{X}$. Recall from the eigendecomposition we have $\mathbf{X} = \sum_{i=1}^n \lambda_i \mathbf{v}_i\mathbf{v}_i^\top$ where we have fixed each $\mathbf{v}_i \in [0,1]^n$ through a sigmoid. Using the scalar SFE $p_{\mathbf{x}}$ we may write each $\mathbf{v}_i$ as a convex combination $\mathbf{v}_i = \sum_S p_{\mathbf{v}_i}(S)\mathbf{1}_S$. For each i we may use this representation to re-express the outer product of $\mathbf{v}_i$ with itself:

$$\begin{aligned}\mathbf{v}_i\mathbf{v}_i^\top &= \big(\sum_S p_{\mathbf{v}_i}(S)\mathbf{1}_S\big)\big(\sum_T p_{\mathbf{v}_i}(T)\mathbf{1}_T\big)^\top \\ &= \sum_S p_{\mathbf{v}_i}(S)^2\mathbf{1}_S\mathbf{1}_S^\top + \sum_{S\neq T} p_{\mathbf{v}_i}(S)p_{\mathbf{v}_i}(T)(\mathbf{1}_T\mathbf{1}_S^\top + \mathbf{1}_S\mathbf{1}_T^\top)\end{aligned}$$

Summing over all eigenvectors $\mathbf{v}_i$ yields the relation $\mathbf{X} = \sum_{S,T\subseteq[n]} p_{\mathbf{X}}(S,T)(\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top)$, proving dual SDP feasibility.

Next, consider an input $\mathbf{X} = \mathbf{1}_S \mathbf{1}_S^\top$. In this case, the only eigenvector is $\mathbf{1}_S$ with eigenvalue $\lambda = |S|$ since $\mathbf{X}\mathbf{1}_S = \mathbf{1}_S(\mathbf{1}_S^\top \mathbf{1}_S) = \mathbf{1}_S|S|$. That is, $p_{\mathbf{X}}(T', T) = p_{\mathbf{1}_S}(T')p_{\mathbf{1}_S}(T)$.

For $\mathbf{X} = \mathbf{1}_S\mathbf{1}_S^\top$, $\mathbf{1}_S$ is clearly an eigenvector with eigenvalue $\lambda = |S|$ because $\mathbf{X}\mathbf{1}_S = \mathbf{1}_S(\mathbf{1}_S^\top \mathbf{1}_S) = \mathbf{1}_S|S|$. So, taking $\bar{\mathbf{1}}_S = \mathbf{1}_S/\sqrt{|S|}$ to be the normalized eigenvector of $\mathbf{X}$, we have $\mathbf{X} = |S|\bar{\mathbf{1}}_S\bar{\mathbf{1}}_S^\top = |S|\left(\frac{\mathbf{1}_S}{\sqrt{|S|}}\right)\left(\frac{\mathbf{1}_S}{\sqrt{|S|}}\right)^\top = p_{\mathbf{X}}(S,S)\mathbf{1}_S\mathbf{1}_S^\top$ for $p_{\mathbf{X}}(S,S) = 1$. Therefore, the corresponding neural SFE is

$$\mathfrak{F}(\mathbf{1}_S\mathbf{1}_S^\top) = p_{\mathbf{X}}(S,S)f(S \cap S) = f(S).$$

All that remains is to show continuity of neural SFEs. Since the scalar SFE $p_{\mathbf{x}}$ is continuous in $\mathbf{x}$ by assumption, all that remains is to show that the map sending $\mathbf{X}$ to its eigenvector with i-th largest eigenvalue is continuous. We handle sign flip invariance of eignevectors by assuming a standard choice for eigenvector signs—e.g., by flipping the sign where necessary to ensure that the first non-zero coordinate is greater than zero. The continuity of the mapping $\mathbf{X} \mapsto \mathbf{v}_i$ follows directly from Theorem 2 from Yu et al. (2015), which is a variant of the Davis–Kahan theorem. The result shows that the angle between the i-th eigenspaces of two matrices $\mathbf{X}$ and $\mathbf{X}'$ goes to zero in the limit as $\mathbf{X} \to \mathbf{X}'$. □

B.2.2 The cross term lemma

Lemma 5. Let V be a ground set of n elements. Let $S_i, S_j \subseteq V$, and $\mathbf{1}_{S_i}, \mathbf{1}_{S_j}$ be their corresponding characteristic vectors. We have

$$\mathbf{1}_{S_i}\mathbf{1}_{S_j}^\top + \mathbf{1}_{S_j}\mathbf{1}_{S_i}^\top = \mathbf{1}_{U_{ij}}\mathbf{1}_{U_{ij}}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top - (\mathbf{1}_{S_j\setminus I_{ij}})(\mathbf{1}_{S_j\setminus I_{ij}})^\top - (\mathbf{1}_{S_i\setminus I_{ij}})(\mathbf{1}_{S_i\setminus I_{ij}})^\top, \tag{B.5}$$

where $U_{ij} = S_i \bigcup S_j, I_{ij} = S_i \bigcap S_j$.

Proof. First, note that The claim of the lemma can then be obtained by carefully expanding the terms on its right-hand side. We group the terms in the following manner

$$\mathbf{1}_{S_i}\mathbf{1}_{S_j}^\top + \mathbf{1}_{S_j}\mathbf{1}_{S_i}^\top = \underbrace{\mathbf{1}_{U_{ij}}\mathbf{1}_{U_{ij}}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top}_{A} - \underbrace{((\mathbf{1}_{S_j} - \mathbf{1}_{I_{ij}})(\mathbf{1}_{S_j} - \mathbf{1}_{I_{ij}})^\top + (\mathbf{1}_{S_i} - \mathbf{1}_{I_{ij}})(\mathbf{1}_{S_i} - \mathbf{1}_{I_{ij}})^\top)}_{B}. \tag{B.6}$$

Starting with term A, from the definition of the union $\mathbf{1}_{U_{ij}} = \mathbf{1}_{S_i} + \mathbf{1}_{S_j} - \mathbf{1}_{I_{ij}}$, we have

$$\begin{aligned}\mathbf{1}_{U_{ij}}\mathbf{1}_{U_{ij}}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top &= (\mathbf{1}_{S_i} + \mathbf{1}_{S_j} - \mathbf{1}_{I_{ij}})(\mathbf{1}_{S_i} + \mathbf{1}_{S_j} - \mathbf{1}_{I_{ij}})^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top \\ &= (\mathbf{1}_{S_i}\mathbf{1}_{S_i}^\top + \mathbf{1}_{S_i}\mathbf{1}_{S_j}^\top - \mathbf{1}_{S_i}\mathbf{1}_{I_{ij}}^\top) + (\mathbf{1}_{S_j}\mathbf{1}_{S_i}^\top + \mathbf{1}_{S_j}\mathbf{1}_{S_j}^\top - \mathbf{1}_{S_j}\mathbf{1}_{I_{ij}}^\top) + (-\mathbf{1}_{I_{ij}}\mathbf{1}_{S_i}^\top - \mathbf{1}_{I_{ij}}\mathbf{1}_{S_j}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top) + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top \\ &= (\mathbf{1}_{S_i}\mathbf{1}_{S_i}^\top + \mathbf{1}_{S_j}\mathbf{1}_{S_j}^\top) + (\mathbf{1}_{S_i}\mathbf{1}_{S_j}^\top + \mathbf{1}_{S_j}\mathbf{1}_{S_i}^\top) - (\mathbf{1}_{S_i}\mathbf{1}_{I_{ij}}^\top + \mathbf{1}_{S_j}\mathbf{1}_{I_{ij}}^\top) - (\mathbf{1}_{I_{ij}}\mathbf{1}_{S_i}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{S_j}^\top) + 2(\mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top).\end{aligned} \tag{B.7}$$

Expanding term B we obtain

$$\begin{aligned}
&(\mathbf{1}_{S_j} - \mathbf{1}_{I_{ij}})(\mathbf{1}_{S_j} - \mathbf{1}_{I_{ij}})^\top + (\mathbf{1}_{S_i} - \mathbf{1}_{I_{ij}})(\mathbf{1}_{S_i} - \mathbf{1}_{I_{ij}})^\top = \\
&= (\mathbf{1}_{S_j}\mathbf{1}_{S_j}^\top - \mathbf{1}_{S_j}\mathbf{1}_{I_{ij}}^\top - \mathbf{1}_{I_{ij}}\mathbf{1}_{S_j}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top) + (\mathbf{1}_{S_i}\mathbf{1}_{S_i}^\top - \mathbf{1}_{S_i}\mathbf{1}_{I_{ij}}^\top - \mathbf{1}_{I_{ij}}\mathbf{1}_{S_i}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top) \\
&= (\mathbf{1}_{S_i}\mathbf{1}_{S_i}^\top + \mathbf{1}_{S_j}\mathbf{1}_{S_j}^\top) - (\mathbf{1}_{S_i}\mathbf{1}_{I_{ij}}^\top + \mathbf{1}_{S_j}\mathbf{1}_{I_{ij}}^\top) - (\mathbf{1}_{I_{ij}}\mathbf{1}_{S_i}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{S_j}^\top) + 2(\mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top) \\
&\overset{B.7}{=} \mathbf{1}_{U_{ij}}\mathbf{1}_{U_{ij}}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top - (\mathbf{1}_{S_i}\mathbf{1}_{S_j}^\top + \mathbf{1}_{S_j}\mathbf{1}_{S_i}^\top). \qquad \text{(B.8)}
\end{aligned}$$

Returning to the righthand side of equation B.6 we have

$$\begin{aligned}
&\mathbf{1}_{U_{ij}}\mathbf{1}_{U_{ij}}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top - ((\mathbf{1}_{S_j} - \mathbf{1}_{I_{ij}})(\mathbf{1}_{S_j} - \mathbf{1}_{I_{ij}})^\top + (\mathbf{1}_{S_i} - \mathbf{1})(\mathbf{1}_{S_i} - \mathbf{1}_{I_{ij}})^\top) \\
&\overset{B.8}{=} \mathbf{1}_{U_{ij}}\mathbf{1}_{U_{ij}}^\top + \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top - \mathbf{1}_{U_{ij}}\mathbf{1}_{U_{ij}}^\top - \mathbf{1}_{I_{ij}}\mathbf{1}_{I_{ij}}^\top + (\mathbf{1}_{S_i}\mathbf{1}_{S_j}^\top + \mathbf{1}_{S_j}\mathbf{1}_{S_i}^\top) \\
&= \mathbf{1}_{S_j}\mathbf{1}_{S_i}^\top + \mathbf{1}_{S_i}\mathbf{1}_{S_j}^\top.
\end{aligned}$$

which concludes the proof. □

B.3 Primal and dual LP and SDP derivations

Derivation of the dual LP.

$$\max_{\mathbf{z},b\in\mathbb{R}^n\times\mathbb{R}} \{\mathbf{x}^\top\mathbf{z} + b\} \text{ subject to } \mathbf{1}_S^\top\mathbf{z} + b \le f(S) \text{ for all } S \subseteq [n].$$

The dual is

$$\min_{\{y_S\ge 0\}_{S\subseteq[n]}} \sum_{S\subseteq[n]} y_S f(S) \text{ subject to } \sum_{S\subseteq[n]} y_S\mathbf{1}_S = \mathbf{x}, \ \sum_{S\subseteq[n]} y_S = 1, \text{ for all } S \subseteq [n].$$

In order to standardize the derivation, we first convert the primal maximization problem into minimization (this will be undone at the end of the derivation). We have

$$\min_{\mathbf{z},b\in\mathbb{R}^n\times\mathbb{R}} \{-\mathbf{x}^\top\mathbf{z} - b\} \text{ subject to } \mathbf{1}_S^\top\mathbf{z} + b \le f(S) \text{ for all } S \subseteq [n]. \qquad \text{(B.9)}$$

The Lagrangian is

$$\begin{aligned}
\underset{y_S\ge 0}{\mathcal{L}(\mathbf{z}, y_S, b)} &= -\mathbf{x}^\top\mathbf{z} - b - \sum_{S\subseteq[n]} y_S(f(S) - \mathbf{1}_S^\top\mathbf{z} - b) \qquad \text{(B.10)} \\
&= -\sum_{S\subseteq[n]} y_S f(S) + (\sum_{S\subseteq[n]} y_S\mathbf{1}_S^\top - \mathbf{x}^\top)\mathbf{z} + b(\sum_{S\subseteq[n]} y_S - 1) \qquad \text{(B.11)}
\end{aligned}$$

The optimal solution $\mathbf{p}^*$ to the primal problem is then

$$\begin{aligned}\mathbf{p}^* &= \min_{\mathbf{z},b} \max_{y_S \geq 0} \mathcal{L}(\mathbf{z}, y_S, b) && \text{(B.12)}\\ &= \max_{y_S \geq 0} \min_{\mathbf{z},b} \mathcal{L}(\mathbf{z}, y_S, b) && \text{(strong duality)}\\ &= \mathbf{d}^*, && \text{(B.13)}\end{aligned}$$

where $\mathbf{d}^*$ is the optimal solution to the dual. From the Lagrangian,

$$\min_{\mathbf{z},b} \mathcal{L}(\mathbf{z}, y_S, b) = \begin{cases} -\sum_{S\subseteq[n]} y_S f(S), \text{ if } \sum_{S\subseteq[n]} y_S \mathbf{1}_S = \mathbf{x} \text{ and } \sum_{S\subseteq[n]} y_S = 1, \\ -\infty, \text{ otherwise.} \end{cases} \tag{B.14}$$

Thus, we can write the dual problem as

$$\mathbf{d}^* = \max_{y_S \geq 0} - \sum_{S\subseteq[n]} y_S f(S) \text{ subject to } \sum_{S\subseteq[n]} y_S \mathbf{1}_S = \mathbf{x} \text{ and } \sum_{S\subseteq[n]} y_S = 1. \tag{B.15}$$

Our proposed dual formulation is then obtained by switching from maximization to minimization and negating the objective. It can also be verified that by taking the dual of our dual, the primal is recovered (see El Halabi (2018, Def. 20) for the derivation).

Derivation of the dual SDP. The dual of our primal SDP can be straightforwardly obtained by following the standard conventions. To facilitate the discussion that will follow, we provide a complete derivation first. Recall that our primal SDP is defined as

$$\max_{\mathbf{Z}\succeq 0, b\in\mathbb{R}} \{\mathrm{Tr}(\mathbf{X}^\top \mathbf{Z}) + b\} \text{ subject to } \frac{1}{2}\mathrm{Tr}((\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top)\mathbf{Z}) + b \leq f(S\cap T) \text{ for } S, T \subseteq [n]. \tag{B.16}$$

We will show that the dual is

$$\min_{\{y_{S,T}\geq 0\}} \sum_{S,\subseteq[n]} y_{S,T} f(S\cap T) \text{ subject to } \mathbf{X} \preceq \sum_{S,T\subseteq[n]} \frac{1}{2} y_{S,T}(\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top) \text{ and } \sum_{S,T\subseteq[n]} y_{S,T} = 1. \tag{B.17}$$

As before, we convert the primal to a minimization problem:

$$\max_{\mathbf{Z}\succeq 0, b\in\mathbb{R}} \{-\mathrm{Tr}(\mathbf{X}^\top \mathbf{Z}) - b\} \text{ subject to } \frac{1}{2}\mathrm{Tr}((\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top)\mathbf{Z}) + b \leq f(S\cap T) \text{ for } S, T \subseteq [n]. \tag{B.18}$$

First, we will standardize the formulation by converting the inequality constraints into equality constraints. This can be achieved by adding a positive slack variable $d_{S,T}$ to each constraint such that

$$\frac{1}{2}\mathrm{Tr}((\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top)\mathbf{Z}) + b + d_{S,T} = f(S\cap T). \tag{B.19}$$

In matrix notation this is done by introducing the positive diagonal slack matrix **D** to the decision variable **Z**, and extending the symmetric matrices in each constraint

$$\mathbf{Z}' = \begin{bmatrix} \mathbf{Z} & 0 \\ 0 & \mathbf{D} \end{bmatrix}, \quad \mathbf{X}' = \begin{bmatrix} \mathbf{X} & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{A}'_{S,T} = \begin{bmatrix} \frac{1}{2}(\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top) & 0 \\ 0 & \operatorname{diag}(\mathbf{e}_{S,T}) \end{bmatrix}, \tag{B.20}$$

where $\operatorname{diag}(\mathbf{e}_{S,T})$ is a diagonal matrix where all diagonal entries are zero except at the diagonal entry corresponding to the constraint on S, T which has a 1. Using this reformulation, we obtain an equivalent SDP in standard form:

$$\max_{\mathbf{Z}'\succeq 0, b\in\mathbb{R}} \{-\operatorname{Tr}(\mathbf{X}'^\top\mathbf{Z}') - b\} \text{ subject to } \operatorname{Tr}(\mathbf{A}'_{S,T}\mathbf{Z}') + b = f(S\cap T) \text{ for } S, T \subseteq [n]. \tag{B.21}$$

Next, we form the Lagrangian which features a decision variable $y_{S,T}$ for each inequality, and a dual matrix variable $\mathbf{\Lambda}$. We have

$$\mathcal{L}(\mathbf{Z}', b, y_{S,T}, \mathbf{\Lambda}) = -\operatorname{Tr}(\mathbf{X}'^\top\mathbf{Z}') - b - \sum_{S,T\subseteq[n]} y_{S,T}\left(2f(S\cap T) - \operatorname{Tr}(\mathbf{A}'_{S,T}\mathbf{Z}') - b\right) - \operatorname{Tr}(\mathbf{\Lambda}\mathbf{Z}') \tag{B.22}$$

$$= \operatorname{Tr}\left(((\sum_{S,T\subseteq[n]} y_{S,T}\mathbf{A}'_{S,T}) - \mathbf{X}' - \mathbf{\Lambda})\mathbf{Z}' \right) + b(\sum_{S,T\subseteq[n]} y_{S,T} - 1) - \sum_{S,T\subseteq[n]} y_{S,T} f(S\cap T) \tag{B.23}$$

For the solution to the primal $\mathbf{p}^*$, we have

$$\begin{aligned} \mathbf{p}^* &= \min_{\mathbf{Z}',b}\max_{\mathbf{\Lambda},y_{S,T}} \mathcal{L}(\mathbf{Z}', b, y_{S,T}, \mathbf{\Lambda}) && \text{(B.24)} \\ &\geq \max_{\mathbf{\Lambda},y_{S,T}}\min_{\mathbf{Z}',b} \mathcal{L}(\mathbf{Z}', b, y_{S,T}, \mathbf{\Lambda}) && \text{(weak duality)} \\ &= \mathbf{d}^*. && \text{(B.25)} \end{aligned}$$

For our Lagrangian we have the dual function

$$\min_{\mathbf{Z}',b} \mathcal{L}(\mathbf{Z}', b, y_{S,T}, \mathbf{\Lambda}) = \begin{cases} 0, \text{ if } \mathbf{\Lambda} \succeq 0, \\ -\infty, \text{ otherwise}. \end{cases} \tag{B.26}$$

Thus, the dual function $\min_{\mathbf{Z}',b} \mathcal{L}(\mathbf{Z}', b, y_{S,T}, \mathbf{\Lambda})$ takes non-infinite values under the conditions

$$\begin{aligned} (\sum_{S,T\subseteq[n]} y_{S,T}\mathbf{A}'_{S,T}) - \mathbf{X}' - \mathbf{\Lambda} = 0, && \text{(B.27)} \\ \mathbf{\Lambda} \succeq 0, && \text{(B.28)} \\ \text{and } \sum_{S,T\subseteq[n]} y_{S,T} - 1 = 0. && \text{(B.29)} \end{aligned}$$

The first two conditions imply the linear matrix inequality (LMI)

$$\sum_{S,T\subseteq[n]} y_{S,T}\mathbf{A}'_{S,T} - \mathbf{X}' \succeq 0. \tag{$\boldsymbol{\Lambda}\succeq 0$}$$

From the definition of $\mathbf{A}'_{S,T}$ we know that its additional diagonal entries will correspond to the variables $y_{S,T}$. Combined with the conditions above, we arrive at the constraints of the dual

$$y_{S,T} \geq 0, \tag{B.30}$$

$$\sum_{S,T\subseteq[n]} \frac{1}{2} y_{S,T}(\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top) \succeq \mathbf{X}, \tag{B.31}$$

$$\sum_{S,T\subseteq[n]} y_{S,T} = 1. \tag{B.32}$$

This leads us to the dual formulation

$$\max_{y_{S,T}\geq 0} -\sum_{S,T\subseteq[n]} y_{S,T} f(S\cap T) \text{ subject to } \sum_{S,T\subseteq[n]} \frac{1}{2} y_{S,T}(\mathbf{1}_S\mathbf{1}_T^\top + \mathbf{1}_T\mathbf{1}_S^\top) \succeq \mathbf{X} \text{ and } \sum_{S,T\subseteq[n]} y_{S,T} = 1. \tag{B.33}$$

Then, we can obtain our original dual by switching to minimization and negating the objective.

B.4 General Experimental Background Information

B.4.1 Hardware and Software Setup

All training runs were done on a single GPU at a time. Experiments were either run on 1) a server with 8 NVIDIA RTX 2080 Ti GPUs, or 2) 4 NVIDIA RTX 2080 Ti GPUs. All experiments are run using Python, specifically the PyTorch (Paszke et al., 2019) framework (see licence here). For GNN specific functionality, such as graph data batching, use the PyTorch Geometric (PyG) (Fey and Lenssen, 2019) (MIT License).

We shall open source our code with MIT License, and have provided anonymized code as part of the supplementary material for reviewers.

B.4.2 Data Details

This paper uses five graph datasets: ENZYMES, PROTEINS, IMDB-BINARY, MUTAG, and COLLAB. All data is accessed via the standardized PyG API. In the case of COLLAB, which has 5000 samples available, we subsample the first 1000 graphs only for training efficiency. All experiments Use a train/val/test split ratio of 60/30/10, which is done in exactly one consistent way across all experiments for each dataset.

B.5 Unsupervised Neural Combinatorial Optimization Experiments

All methods use the same GNN backbone: a combination of GAT Veličković et al. (2018) and Gated Graph Convolution layer (Yujia et al., 2016). We use the Adam optimizer Kingma and Ba (2014) with initial $lr = 10^{-4}$ and default PyTorch settings for other parameters Paszke et al. (2019). We use grid search HPO over batch size $\{4, 32, 64\}$, number of GNN layers $\{6, 10, 16\}$ network width $\{64, 128, 256\}$. All models are trained for 200 epochs. For the model with the best validation performance, we report the test performance and the standard deviation of performance over test graphs as a measure of method reliability.

B.5.1 Discrete Objectives

Maximum Clique. For the maximum clique problem, we could simply take f to compute the clique size (with the size being zero if S is not a clique). However, we found that this objective led to unstable training dynamics. So, instead, we select a discrete objective that yielded the much more stable results across datasets. It is defined for a graph $G = ([n], E)$ as,

$$f_{\text{MaxClique}}(S; G) = w(S) q^c(S), \tag{B.34}$$

where w is a measure of size of S and q measures the density of edges within S (i.e., distance from being a clique). The scalar c is a constant, taken to be $c = 2$ in all cases except REINFORCE for which $c = 2$ proved ineffective, so we use $c = 4$ instead. Specifically, $w(S) = \sum_{i,j \in S} \mathbf{1}\{(i, j) \in E\}$ simply counts up all the edges between nodes in S, and $q(S) = -2w(S)/(|S|^2 - |S|)$ is the ratio (with a sign flip) between the number of edges in S, and the number of undirected edges $(|S|^2 - |S|)/2$ there would be in a clique of size $|S|$. If G were directed, simply remove the factor of 2.

Maximum Independent Set. Similarly for maximum independent set we use the discrete objective,

$$f_{\text{MIS}}(S; G) = w(S) q^c(S), \tag{B.35}$$

where w is a measure of size of S and q measures the number of edges between nodes in S (the number should be zero for an independent set), and $c = 2$ as before. Specifically, we take $w(S) = |S|/n$, and $q(s) = 2\sum_{i,j \in S} \mathbf{1}\{(i, j) \in E\}/(|S|^2 - |S|)$, as before.

B.5.2 Neural SFE details.

All Neural SFEs, unless otherwise stated, use the top $k = 4$ eigenvectors corresponding to the largest eigenvalues. This is an important efficiency saving step, since with $k = n$, i.e., using all eigenvectors, the resulting Neural Lovaśz extension requires $O(n^2)$ set function evaluations, compared to $O(n)$ for the scalar Lovaśz extension. By only using the top k we reduce the number of evaluations to $O(kn)$. Wall clock runtime experiments given in Figure 3.4 show that the runtime of the Neural Lovaśz extension is around $\times k$ its scalar counterpart, and that the performance of the neural extension gradually increases then saturates when k gets large. To minimize compute overheads we pick the smallest k at which performance saturation approximately occurs.

Instead of calling the pre-implemented PyTorch eigensolver `torch.linalg.eigh`, which calls LAPACK routines, we use the power method to approximate the first k eignevectors of $\mathbf{X}$. This is because we found the PyTorch function to be too numerically unstable in our case. In contrast, we found the power method, which approximates eigenvectors using simple recursively defined polynomials of $\mathbf{X}$, to be significantly more reliable. In all cases we run the power method for 5 iterations, which we found to be sufficient for convergence.

B.5.3 Baselines.

This section discusses various implementation details of the baseline methods we used. The basic training pipeline is kept identical to SFEs, unless explicitly said otherwise. Namely, we use nearly identical model architectures, identical data loading, and identical HPO parameter grids.

REINFORCE. We compared with REINFORCE (Williams (1992)) which enables backpropagation through (discrete) black-box functions. We opt for a simple instantiation for the score estimator

$$\hat{g}_{\text{REINFORCE}} = f(S)\frac{\partial}{\partial\theta}\log p(S|\theta), \tag{B.36}$$

where $p(S|\theta) = \prod_{i\in S} p_i \prod_{j\notin S}(1-p_j)$, i.e., each node is selected independently with probability $p_i = g_\theta(\mathbf{y})$ for $i = 1,2,\ldots,n$, where g_θ is a neural network and $\mathbf{y}$ some input attributes. We maximize the expected reward, i.e.,

$$L_{\text{REINFORCE}}(\theta) = \mathbb{E}_{S\sim\theta}[\hat{g}_{\text{REINFORCE}}]. \tag{B.37}$$

For all experiments with REINFORCE, the expected reward is computed over 250 sampled actions S which is approximately the number of function evaluations of neural SFEs in most of the datasets. Here, f is taken to be the corresponding discrete objective of each problem (as described earlier in section B.5.1). For maximum clique, we normalize rewards $f(S)$ by removing the mean and dividing by the standard deviation. For the maximum independent set, the same strategy led to severe instability during training. To alleviate the issue, we introduced an additional modification to the rewards: among the sampled actions S, only the ones that achieved higher than average reward were retained and the rewards of the rest were set to 0. This led to more stable results in most datasets, with the exception of COLLAB were the trick was not sufficient.

These issues highlight the instability of the score function estimator in this kind of setting. Additionally, we experimented by including simple control variates (baselines). These were: i) a simple greedy baseline obtained by running a greedy algorithm on each input graph ii) a simple uniform distribution baseline, where actions S were sampled uniformly at random. Unfortunately, we were not able to obtain any consistent boost in either performance or stability using those techniques. Finally, to improve stability, the architectures employed with REINFORCE were slightly modified according to the problem. For example, for the independent set we additionally applied a sigmoid to the outputs of the final layer.

Erdos Goes Neural. We compare with recent work on unsupervised combinatorial optimization (Karalias and Loukas, 2020). We use the probabilistic methodology described in the paper to obtain a loss function for each problem. For the MaxClique, we use the loss provided in the paper, where for an input graph $G = ([n], E)$ and learned probabilities $\mathbf{p}$ it is calculated by

$$L_{\text{Clique}}(\mathbf{p}; G) = (\beta + 1) \sum_{(i,j)\in E} w_{ij} p_i p_j + \frac{\beta}{2} \sum_{v_i \neq v_j} p_i p_j. \tag{B.38}$$

We omit additive constants as in practice they not affect the optimization. For the maximum independent set, we follow the methodology from the paper to derive the following loss:

$$L_{\text{IndepSet}}(\mathbf{p}; G) = \beta \sum_{(i,j)\in E} w_{ij} p_i p_j - \sum_{v_i \in V} p_i. \tag{B.39}$$

β was tuned through a simple line search over a few possible values in each case. Following the implementation of the original paper, we use the same simple decoding algorithm to obtain a discrete solution from the learned probabilities.

Straight Through Estimator. We also compared with the Straight-Through gradient estimator (Bengio et al., 2013). This estimator can be used to pass gradients through sampling and thresholding operations, by assuming in the backward pass that the operation is the identity. In order to obtain a working baseline with the straight-through estimator, we generate level sets according to the ranking of elements in the output vector $\mathbf{x}$ of the neural network. Specifically, given $\mathbf{x} \in [0,1]^n$ outputs from a neural network, we generate indicator vectors $\mathbf{1}_{S_k}$, where $S_k = \{j \mid x_j \geq x_k\}$ for $k = 1, 2, \ldots, n$. Then our loss function was computed as

$$L_{ST}(\mathbf{x}; G) = \frac{1}{n} \sum_{k=1}^{n} f(\mathbf{1}_{S_k}), \tag{B.40}$$

where f is the corresponding discrete objective from section B.5.1. At inference, we select the set that achieves the best value in the objective while complying with the constraints.

Ground truths. We obtain the maximum clique size and the maximum independent set size s for each graph by expressing it as a mixed integer program and using the Gurobi solver

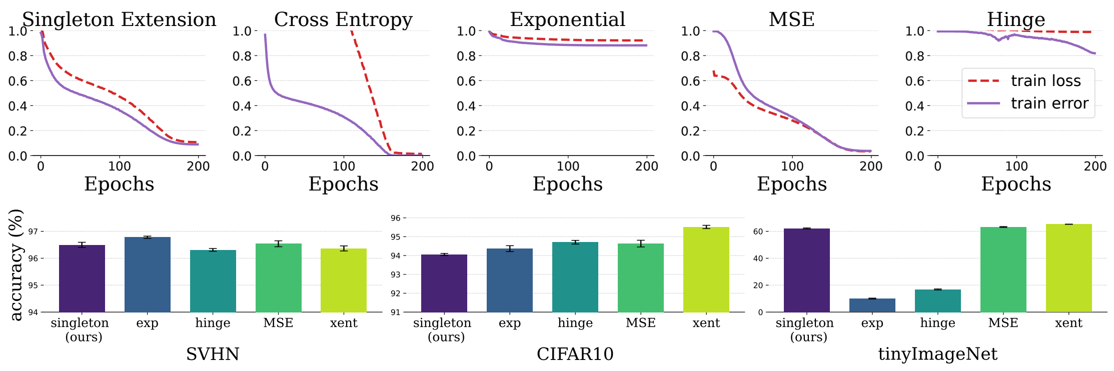


Figure B.1: Top: Additional experimental results on the tinyImageNet dataset. Bottom: test accuracies of different losses. The singleton extension performs broadly comparably to other losses.

(Gurobi Optimization, 2020).

B.5.4 k-Clique Constraint Satisfaction

Ground truths. As before, we obtain the maximum clique size s for each graph by expressing it as a mixed integer program and using the Gurobi solver (Gurobi Optimization, 2020). This is converted into a binary label $\mathbf{1}\{s \geq k\}$ indicating if there is a clique of size k or bigger.

Implementation details. The training pipeline, including HPO, is identical to the MaxClique setup. The only difference comes in the evaluation—at test time the GNN produces an embedding $\mathbf{x}$, and the largest clique S in the support of $p_{\mathbf{x}}$ is selected. The model prediction for the constraint satisfaction problem is then $\mathbf{1}\{|S| \geq k\}$, indicating whether the GNN found a clique of size k or more. Since this problem is. binary classification problem we compute the F1-score on a validation set, and report as the final result the F1-score of that same model on the test set.

B.6 Training error as an objective

Recall that for a K-way classifier $h : \mathcal{X} \to \mathbb{R}^K$ with $\hat{y}(x) = \arg\max_{k=1,\ldots,K} h(x)_k$, we consider the training error $\frac{1}{n}\sum_{i=1}^n \mathbf{1}\{y_i \neq \hat{y}(x_i)\}$ calculated over a labeled training dataset $\{(x_i, y_i)\}_{i=1}^n$ to be a discrete non-differentiable loss. The set function in question is $y \mapsto \mathbf{1}\{y_i \neq y\}$, which we relax using the singleton method described in Section 3.3.1.

Training details. For all datasests we use a standard ResNet-18 backbone, with a final layer to output a vector of the correct dimension depending on the number of classes in the dataset. CIFAR10 and tinyImageNet models are trained for 200 epochs, while SVHN uses 100 (which is sufficient for convergence). We use SGD with momentum $mom = 0.9$ and weight decay $wd = 5 \times 10^{-4}$ and a cosine learning rate schedule. We tune the learning rate for each loss via a simple grid search of the values $lr \in \{0.01, 0.05, 0.1, 0.2\}$. For each loss we select the learning rate with highest accuracy on a validation set, then display the training loss and accuracy for this run.

B.7 Pseudocode: A forward pass of Scalar and Neural SFEs

To illustrate the main conceptual steps in the implementation of SFEs, we include two torch-like pseudocode examples for SFEs, one for scalar and one for neural SFEs. The key to the practical implementation of SFEs within PyTorch is that it is only necessary to define the forward pass. Gradients are then handled automatically during the backwards pass.

Observe that in both Algorithm, 1 and Algorithm 2, there are two key functions that have to be implemented: i) getSupportSets, which generates the sets on which the extension is supported. ii) getCoeffs, which generates the coefficients of each set. Those depend on the choice of the extension and have to be implemented from scratch whenever a new extension is designed. The sets of the neural extension and their coefficients can be calculated from the corresponding scalar ones, using the definition of the Neural SFE and Proposition 3.

Algorithm 1: Scalar set function extension

```
def ScalarSFE(setFunction, x):
    # x:  n x 1 tensor of embeddings, the output of a neural network
    # n:  number of items in ground set (e.g.  number of nodes in
     graph)
    setsScalar = getSupportSetsScalar(x) # n x n, i-th column is $S_i$.
    coeffsScalar = getCoeffsScalar(x) # 1 x n:  coefficients $y_{S_i}$.
    extension = (coeffsScalar*setFunction(setsScalar)).sum()
    return extension
```

Algorithm 2: Neural set function extension

```
def NeuralSFE(setFunction, X):
    # X: n x d tensor of embeddings, the output of a neural network
    # n:  number of items in ground set (e.g.  number of nodes in
     graph)
    # d:  embedding dimension
    X = normalize(X, dim=1)
    Gram = X @ X.T #  n x n
    eigenvalues, eigenvectors = powerMethod(Gram)
    extension = 0 # initialize variable
    for (eigval,eigvec) in zip(eigenvalues,eigenvectors):
        # Compute scalar extension data.
        setsScalar = getSupportSetsScalar(eigvec)
        coeffsScalar = getCoeffsScalar(eigvec)
        # Compute neural extension data from scalar extension data.
        setsNeural = getSupportSetsNeural(setsScalar)
        coeffsNeural = getCoeffsNeural(coeffsScalar)
        extension +=
         eigval*((coeffsNeural*setFunction(setsNeural)).sum())
    return extension
```

BIBLIOGRAPHY

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P., Cao, X., Khosravi, A., Acharya, U. R., et al. (2021). A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297.

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. (2019). Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 32:9562–9574.

Alon, N. and Spencer, J. H. (2004). *The probabilistic method*. John Wiley & Sons.

Alon, N. and Spencer, J. H. (2016). *The probabilistic method*. John Wiley & Sons.

Amizadeh, S., Matusevych, S., and Weimer, M. (2018). Learning to solve circuit-sat: An unsupervised differentiable approach.

Amizadeh, S., Matusevych, S., and Weimer, M. (2019). Pdp: A general neural framework for learning constraint satisfaction solvers.

Amos, B. and Kolter, J. Z. (2017). Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR.

Andersen, R., Chung, F., and Lang, K. (2006). Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE.

Andreev, K. and Racke, H. (2006). Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939.

Ansótegui, C., Giráldez-Cru, J., and Levy, J. (2012). The community structure of sat formulas. In *Theory and Applications of Satisfiability Testing–SAT 2012: 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings 15*, pages 410–423. Springer.

Arakelyan, E., Daza, D., Minervini, P., and Cochez, M. (2020). Complex query answering with neural link predictors. *arXiv preprint arXiv:2011.03459*.

Ardila, F. (2021). The geometry of geometries: matroid theory, old and new. *arXiv preprint arXiv:2111.08726*.

Ardila, F., Benedetti, C., and Doker, J. (2010). Matroid polytopes and their volumes. *Discrete & Computational Geometry*, 43(4):841–854.

Bach, F. (2019). Submodular functions: from discrete to continuous domains. *Mathematical Programming*, 175(1):419–459.

Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., and Wang, W. (2018). Graph edit distance computation via graph neural networks. *arXiv preprint arXiv:1808.05689*.

Bai, Y., Xu, D., Wang, A., Gu, K., Wu, X., Marinovic, A., Ro, C., Sun, Y., and Wang, W. (2020). Fast detection of maximum common subgraph via deep q-learning. *arXiv preprint arXiv:2002.03129*.

Barceló, P., Kostylev, E. V., Monet, M., Pérez, J., Reutter, J., and Silva, J. P. (2019). The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., and Bengio, Y. (2012). Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Baxter, J. (2000). A model of inductive bias learning. *Journal of artificial intelligence research*, 12:149–198.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Bengio, Y., Lodi, A., and Prouvost, A. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421.

Bianchi, F. M., Grattarola, D., and Alippi, C. (2019). Mincut pooling in graph neural networks.

Biere, A., Heule, M., and van Maaren, H. (2009). *Handbook of satisfiability*, volume 185. IOS press.

Bilmes, J. (2022). Submodularity in machine learning and artificial intelligence. *arXiv preprint arXiv:2202.00132*.

Birkhoff, G. (1946). Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucuman, Ser. A*, 5:147–154.

Bissacot, R., Fernández, R., Procacci, A., and Scoppola, B. (2011). An improvement of the lovász local lemma via cluster expansion. *Combinatorics, Probability and Computing*, 20(5):709–719.

Bodirsky, M., Kummer, M., and Thom, A. (2022). Spectrahedral shadows and completely positive maps on real closed fields. *arXiv preprint arXiv:2206.06312*.

Boettcher, S. (2023a). Deep reinforced learning heuristic tested on spin-glass ground states: The larger picture. *arXiv preprint arXiv:2302.10848*.

Boettcher, S. (2023b). Inability of a graph neural network heuristic to outperform greedy algorithms in solving combinatorial optimization problems. *Nature Machine Intelligence*, 5(1):24–25.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Bomze, I. M. (1997). Evolution towards the maximum clique. *Journal of Global Optimization*, 10(2):143–164.

Bomze, I. M., Budinich, M., Pardalos, P. M., and Pelillo, M. (1999). The maximum clique problem. In *Handbook of combinatorial optimization*, pages 1–74. Springer.

Boppana, R. and Halldórsson, M. M. (1992). Approximating maximum independent sets by excluding subgraphs. *BIT Numerical Mathematics*, 32(2):180–196.

Böther, M., Kißig, O., Taraz, M., Cohen, S., Seidel, K., and Friedrich, T. (2022). What's wrong with deep learning in tree search for combinatorial optimization. In *International Conference on Learning Representations*.

Bouritsas, G., Frasca, F., Zafeiriou, S., and Bronstein, M. M. (2022). Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668.

Bouritsas, G., Loukas, A., Karalias, N., and Bronstein, M. (2021). Partition and code: learning how to compress graphs. *Advances in Neural Information Processing Systems*, 34:18603–18619.

Brailsford, S. C., Potts, C. N., and Smith, B. M. (1999). Constraint satisfaction problems: Algorithms and applications. *European journal of operational research*, 119(3):557–581.

Bresson, X., Laurent, T., Uminsky, D., and Von Brecht, J. (2013). Multiclass total variation clustering. In *Advances in Neural Information Processing Systems*, pages 1421–1429.

Bruglieri, M., Maffioli, F., and Ehrgott, M. (2004). Cardinality constrained minimum cut problems: complexity and algorithms. *Discrete Applied Mathematics*, 137(3):311–341.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., et al. (2023). Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Calinescu, G., Chekuri, C., Pál, M., and Vondrák, J. (2011). Maximizing a submodular set function subject to a matroid constraint. *SIAM J. Computing*, 40(6).

Cappart, Q., Chételat, D., Khalil, E., Lodi, A., Morris, C., and Veličković, P. (2021a). Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*.

Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., and Veličković, P. (2021b). Combinatorial optimization and reasoning with graph neural networks. In Zhou, Z.-H., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4348–4355. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

Catarata, J. D., Corbett, S., Stern, H., Szegedy, M., Vyskocil, T., and Zhang, Z. (2017). The moser-tardos resample algorithm: Where is the limit?(an experimental inquiry). In *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 159–171. SIAM.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020a). A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

Chen, X. and Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, pages 6278–6289.

Chen, Z., Chen, L., Villar, S., and Bruna, J. (2020b). Can graph neural networks count substructures? *arXiv preprint arXiv:2002.04025*.

Cheng, C.-A., Yan, X., and Boots, B. (2020). Trajectory-wise control variates for variance reduction in policy gradient methods. In *Conference on Robot Learning*, pages 1379–1394. PMLR.

Cheng, M. X., Li, Y., and Du, D.-Z. (2006). *Combinatorial optimization in communication networks*. Springer.

Choquet, G. (1954). Theory of capacities. In *Annales de l'institut Fourier*, volume 5, pages 131–295.

Chung, F. R. and Graham, F. C. (1997). *Spectral graph theory*. Number 92. American Mathematical Soc.

Ciarella, S., Trinquier, J., Weigt, M., and Zamponi, F. (2023). Machine-learning-assisted monte carlo fails at sampling computationally hard problems. *Machine Learning: Science and Technology*, 4(1):010501.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158.

Coulom, R. (2007). Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers 5*, pages 72–83. Springer.

Craven, B. and Mond, B. (1981). Linear programming with matrix variables. *Linear Algebra and its Applications*, 38:73–80.

Dai, H., Chen, X., Li, Y., Gao, X., and Song, L. (2020). A framework for differentiable discovery of graph algorithms.

Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410.

Deac, A.-I., Veličković, P., Milinkovic, O., Bacon, P.-L., Tang, J., and Nikolic, M. (2021). Neural algorithmic reasoners are implicit planners. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15529–15542. Curran Associates, Inc.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., and Rousseau, L.-M. (2018). Learning heuristics for the tsp by policy gradient. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 170–181. Springer.

Dickinson, P. J. and Gijben, L. (2014). On the computational complexity of membership problems for the completely positive cone and its dual. *Computational optimization and applications*, 57:403–415.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. (2018). Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*.

Dudzik, A. and Veličković, P. (2022). Graph neural networks are dynamic programmers. *arXiv preprint arXiv:2203.15544*.

Dufossé, F. and Uçar, B. (2016). Notes on birkhoff–von neumann decomposition of doubly stochastic matrices. *Linear Algebra and its Applications*, 497:108–115.

Dür, M. (2010). Copositive programming–a survey. In *Recent advances in optimization and its applications in engineering*, pages 3–20. Springer.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*.

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467.

Edmonds, J. (2003). Submodular functions, matroids, and certain polyhedra. In *Combinatorial Optimization—Eureka, You Shrink!*, pages 11–26. Springer.

Ehrlich, H.-C. and Rarey, M. (2011). Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(1):68–79.

El Halabi, M. (2018). Learning with structured sparsity: From discrete to convex and back. Technical report, EPFL.

El Halabi, M., Bach, F., and Cevher, V. (2018). Combinatorial penalties: Which structures are preserved by convex relaxations? In *International Conference on Artificial Intelligence and Statistics*, pages 1551–1560. PMLR.

El Halabi, M. and Jegelka, S. (2020). Optimal approximation for unconstrained non-submodular minimization. In *International Conference on Machine Learning*, pages 3961–3972. PMLR.

Emami, P. and Ranka, S. (2018). Learning permutations with sinkhorn policy gradient. *arXiv preprint arXiv:1805.07010*.

Erdös, P. (1959). Graph theory and probability. *Canadian Journal of Mathematics*, 11:34–38.

Falk, J. E. and Hoffman, K. R. (1976). A successive underestimation method for concave minimization problems. *Mathematics of operations research*, 1(3):251–259.

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R Ruiz, F. J., Schrittwieser, J., Swirszcz, G., et al. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53.

Festa, P. (2014). A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems. In *2014 16th International Conference on Transparent Optical Networks (ICTON)*, pages 1–20. IEEE.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*.

Fountoulakis, K., Gleich, D. F., and Mahoney, M. W. (2018). A short introduction to local graph clustering methods and software. *arXiv preprint arXiv:1810.07324*.

Gaile, E., Draguns, A., Ozoliņš, E., and Freivalds, K. (2023). Unsupervised training for neural tsp solver. In *Learning and Intelligent Optimization: 16th International Conference, LION 16, Milos Island, Greece, June 5–10, 2022, Revised Selected Papers*, pages 334–346. Springer.

Ganesh, V. and Vardi, M. Y. (2020). On the unreasonable effectiveness of sat solvers. *Beyond the Worst-Case Analysis of Algorithms*, pages 547–566.

Gao, L., Chen, M., Chen, Q., Luo, G., Zhu, N., and Liu, Z. (2020). Learn to design the heuristics for vehicle routing problem. *arXiv preprint arXiv:2002.08539*.

Garey, M. R., Johnson, D. S., and Stockmeyer, L. (1974). Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63.

Garg, V. K., Jegelka, S., and Jaakkola, T. (2020). Generalization and representational limits of graph neural networks. *arXiv preprint arXiv:2002.06157*.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. (2019). Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549.

Goemans, M. X. (1997). Semidefinite programming in combinatorial optimization. *Mathematical Programming*, 79(1-3):143–161.

Goemans, M. X. and Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145.

Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE.

Goyal, A. and Bengio, Y. (2022). Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A*, 478(2266):20210068.

Grathwohl, W., Choi, D., Wu, Y., Roeder, G., and Duvenaud, D. (2018). Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *International Conference on Learning Representations*.

Grötschel, M., Lovász, L., and Schrijver, A. (1981). The ellipsoid algorithm and its consequences in combinatorial optimization. *Combinatorica*, 1:499–513.

Grötschel, M., Lovász, L., and Schrijver, A. (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197.

Grover, A., Wang, E., Zweig, A., and Ermon, S. (2018). Stochastic optimization of sorting networks via continuous relaxations. In *International Conference on Learning Representations*.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R., and Levine, S. (2017). Q-prop: Sample-efficient policy gradient with an off-policy critic. In *5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings*.

Guessab, A. (2013). Generalized barycentric coordinates and approximations of convex functions on arbitrary convex polytopes. *Computers & Mathematics with Applications*, 66(6):1120–1136.

Gurobi Optimization, L. (2020). Gurobi optimizer reference manual.

Hamilton, W., Bajaj, P., Zitnik, M., Jurafsky, D., and Leskovec, J. (2018). Embedding logical queries on knowledge graphs. *Advances in neural information processing systems*, 31.

Hind, H., Molloy, M., and Reed, B. (1997). Colouring a graph frugally. *Combinatorica*, 17(4):469–482.

Hopfield, J. J. and Tank, D. W. (1985). "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152.

Hormann, K. (2014). Barycentric interpolation. In *Approximation Theory XIV: San Antonio 2013*, pages 197–218. Springer.

Huijben, I. A., Kool, W., Paulus, M. B., and Van Sloun, R. J. (2022). A review of the gumbel-max trick and its extensions for discrete stochasticity in machine learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Iguchi, T., Mixon, D. G., Peterson, J., and Villar, S. (2015). On the tightness of an sdp relaxation of k-means. *arXiv preprint arXiv:1505.04778*.

Irpan, A. (2018). Deep reinforcement learning doesn't work yet. https://www.alexirpan.com/2018/02/14/rl-hard.html.

Iyer, R., Jegelka, S., and Bilmes, J. (2013). Fast semidifferential-based submodular function optimization: Extended version. In *ICML*.

Iyer, R., Jegelka, S., and Bilmes, J. (2014). Monotone closure of relaxed constraints in submodular optimization: Connections between minimization and maximization: Extended version. In *UAI*.

James, J., Yu, W., and Gu, J. (2019). Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 20(10):3806–3817.

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with gumbel-softmax. In *Int. Conf. on Learning Representations (ICLR)*.

johnjforrest, Vigerske, S., Santos, H. G., Ralphs, T., Hafer, L., Kristjansson, B., jpfasano, EdwinStraver, Lubin, M., rlougee, jpgoncal1, h-i gassmann, and Saltzman, M. (2020). coin-or/cbc: Version 2.10.5.

Joshi, C. K., Laurent, T., and Bresson, X. (2019a). An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*.

Joshi, C. K., Laurent, T., and Bresson, X. (2019b). On learning paradigms for the travelling salesman problem.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Tunyasuvunakool, K., Ronneberger, O., Bates, R., Žídek, A., Bridgland, A., et al. (2020). High accuracy protein structure prediction using deep learning. *Fourteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstract Book)*, 22:24.

Karalias, N. and Loukas, A. (2020). Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33:6659–6672.

Karalias, N., Robinson, J. D., Loukas, A., and Jegelka, S. (2022). Neural set function extensions: Learning with discrete functions in high dimensions. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer.

Karpas, E., Abend, O., Belinkov, Y., Lenz, B., Lieber, O., Ratner, N., Shoham, Y., Bata, H., Levine, Y., Leyton-Brown, K., et al. (2022). Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*.

Kennedy, M. C. and O'Hagan, A. (2001). Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464.

Kersting, K., Kriege, N. M., Morris, C., Mutzel, P., and Neumann, M. (2020). Benchmark data sets for graph kernels.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358.

Khalil, E. B., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B. (2016). Learning to branch in mixed integer programming. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Klee, V. and Minty, G. J. (1972). How good is the simplex algorithm. *Inequalities*, 3(3):159–175.

Kolipaka, K. B. R. and Szegedy, M. (2011). Moser and tardos meet lovász. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 235–244.

Kool, W., van Hoof, H., and Welling, M. (2018). Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*.

Koumchatzky, N. and Andryeyev, A. (2017). Using deep learning at scale in twitter's timelines. Accessed on April 23, 2023.

Kramer, O. (2017). *Genetic algorithms*. Springer.

Kumar, V. (1992). Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1):32–32.

Lang, K. and Rao, S. (2004). A flow-based method for improving the expansion or conductance of graph cuts. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 325–337. Springer.

Larkoski, A. J., Moult, I., and Nachman, B. (2020). Jet substructure at the large hadron collider: a review of recent advances in theory and machine learning. *Physics Reports*, 841:1–63.

Lasserre, J. B. (2001). Global optimization with polynomials and the problem of moments. *SIAM Journal on optimization*, 11(3):796–817.

Lee, W., Yu, H., Rival, X., and Yang, H. (2020). On correctness of automatic differentiation for non-differentiable functions. *Advances in Neural Information Processing Systems*, 33:6719–6730.

Lemos, H., Prates, M., Avelar, P., and Lamb, L. (2019). Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data.

Li, Y., Gimeno, F., Kohli, P., and Vinyals, O. (2020). Strong generalization and efficiency in neural programs. *arXiv preprint arXiv:2007.03629*.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Li, Z., Chen, Q., and Koltun, V. (2018). Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pages 539–548.

Lin, M., Murali, V., and Karaman, S. (2022). A planted clique perspective on hypothesis pruning. *IEEE Robotics and Automation Letters*, 7(2):5167–5174.

Liu, H., Feng, Y., Mao, Y., Zhou, D., Peng, J., and Liu, Q. (2018). Action-dependent control variates for policy optimization via stein identity. In *International Conference on Learning Representations*.

Liu, S., Zhang, Y., Tang, K., and Yao, X. (2022). How good is neural combinatorial optimization? *arXiv preprint arXiv:2209.10913*.

Lodi, A. (2010). Mixed integer programming computation. *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 619–645.

Loukas, A. (2019). What graph neural networks cannot learn: depth vs width.

Loukas, A. (2020a). How hard is graph isomorphism for graph neural networks? *arXiv preprint arXiv:2005.06649*.

Loukas, A. (2020b). What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*.

Lovász, L. (1979). On the shannon capacity of a graph. *IEEE Transactions on Information theory*, 25(1):1–7.

Lovász, L. (1983). Submodular functions and convexity. In *Mathematical programming the state of the art*, pages 235–257. Springer.

Lovász, L. (2003). Semidefinite programs and combinatorial optimization. *Recent advances in algorithms and combinatorics*, pages 137–194.

Lovász, L. (2019). *Graphs and geometry*, volume 65. American Mathematical Soc.

Lovász, L., Saks, M., and Schrijver, A. (1989). Orthogonal representations and connectivity of graphs. *Linear Algebra and its applications*, 114:439–454.

Lovász, L. and Schrijver, A. (1991). Cones of matrices and set-functions and 0–1 optimization. *SIAM journal on optimization*, 1(2):166–190.

Maddison, C., Mnih, A., and Teh, Y. (2017). The concrete distribution: A continuous relaxation of discrete random variables. In *Int. Conf. on Learning Representations (ICLR)*.

Marichal, J.-L. (2000). An axiomatic approach of the discrete choquet integral as a tool to aggregate interacting criteria. *IEEE transactions on fuzzy systems*, 8(6):800–807.

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400.

McCarty, E., Zhao, Q., Sidiropoulos, A., and Wang, Y. (2021). Nn-baker: A neural-network infused algorithmic framework for optimization problems on geometric intersection graphs. *Advances in Neural Information Processing Systems*, 34:23023–23035.

Meek, D. (1983). The inverses of toeplitz band matrices. *Linear Algebra and its Applications*, 49:117–129.

Mena, G., Belanger, D., Linderman, S., and Snoek, J. (2018). Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*.

Min, Y., Wenkel, F., Perlmutter, M., and Wolf, G. (2022). Can hybrid geometric scattering networks help solve the maximal clique problem? *arXiv preprint arXiv:2206.01506*.

Mitzenmacher, M. and Upfal, E. (2017). *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. (2020). Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609.

Moser, R. A. and Tardos, G. (2010). A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):1–15.

Motzkin, T. S. and Straus, E. G. (1965). Maxima for graphs and a new proof of a theorem of turán. *Canadian Journal of Mathematics*, 17:533–540.

Murota, K. (1998). Discrete convex analysis. *Mathematical Programming*, 83(1):313–371.

Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9839–9849.

Niculae, V., Corro, C. F., Nangia, N., Mihaylova, T., and Martins, A. F. (2023). Discrete latent structure in neural networks. *arXiv preprint arXiv:2301.07473*.

Niepert, M., Minervini, P., and Franceschi, L. (2021). Implicit mle: Backpropagating through discrete exponential family distributions. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 14567–14579. Curran Associates, Inc.

Nikishin, E., Izmailov, P., Athiwaratkun, B., Podoprikhin, D., Garipov, T., Shvechikov, P., Vetrov, D., and Wilson, A. G. (2018). Improving stability in deep reinforcement learning with weight averaging. In *Uncertainty in Artificial Intelligence Workshop on Uncertainty in Deep Learning*, volume 5.

Nowak, A., Villar, S., Bandeira, A. S., and Bruna, J. (2017). A note on learning algorithms for quadratic assignment with graph neural networks. *stat*, 1050:22.

Nudelman, E., Leyton-Brown, K., Hoos, H. H., Devkar, A., and Shoham, Y. (2004). Understanding random sat: Beyond the clauses-to-variables ratio. In *International Conference on Principles and Practice of Constraint Programming*, pages 438–452. Springer.

Obozinski, G. and Bach, F. (2012). *Convex Relaxation for Combinatorial Penalties*. PhD thesis, INRIA.

Obozinski, G. and Bach, F. (2016). A unified perspective on convex structured sparsity: Hierarchical, symmetric, submodular norms and beyond.

Palm, R., Paquet, U., and Winther, O. (2018). Recurrent relational networks. In *Advances in Neural Information Processing Systems*, pages 3368–3378.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Paulus, A., Rolinek, M., Musil, V., Amos, B., and Martius, G. (2021). Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8443–8453. PMLR.

Paulus, M. B., Choi, D., Tarlow, D., Krause, A., and Maddison, C. J. (2020a). Gradient estimation with stochastic softmax tricks. *arXiv preprint arXiv:2006.08063*.

Paulus, M. B., Maddison, C. J., and Krause, A. (2020b). Rao-blackwellizing the straight-through gumbel-softmax gradient estimator. In *International Conference on Learning Representations*.

Peng, B., Wang, J., and Zhang, Z. (2019). A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In *International Symposium on Intelligence Computation and Applications*, pages 636–650. Springer.

Petersen, F. (2022). Learning with differentiable algorithms. *arXiv preprint arXiv:2209.00616*.

Petersen, F., Borgelt, C., Kuehne, H., and Deussen, O. (2022). Deep differentiable logic gate networks. *arXiv preprint arXiv:2210.08277*.

Potvin, J.-Y. and Gendreau, M. (2018). *Handbook of Metaheuristics*. Springer.

Prates, M., Avelar, P. H., Lemos, H., Lamb, L. C., and Vardi, M. Y. (2019). Learning to solve np-complete problems: A graph neural network for decision tsp. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4731–4738.

Prosser, P. (2012). Exact algorithms for maximum clique: A computational study. *Algorithms*, 5(4):545–587.

Raghavan, P. (1988). Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143.

Ren, H., Hu, W., and Leskovec, J. (2019). Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *International Conference on Learning Representations*.

Rudi, A., Marteau-Ferey, U., and Bach, F. (2020). Finding global minima via kernel approximations. *arXiv preprint arXiv:2012.11978*.

Sanyal, R., Sottile, F., and Sturmfels, B. (2011). Orbitopes. *Mathematika*, 57(2):275–314.

Sato, R. (2020). A survey on the expressive power of graph neural networks.

Sato, R., Yamada, M., and Kashima, H. (2019). Approximation ratios of graph neural networks for combinatorial problems.

Sato, R., Yamada, M., and Kashima, H. (2020). Random features strengthen graph neural networks. *arXiv preprint arXiv:2002.03155*.

Saunderson, J., Parrilo, P. A., and Willsky, A. S. (2015). Semidefinite descriptions of the convex hull of rotation matrices. *SIAM Journal on Optimization*, 25(3):1314–1343.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Scheiderer, C. (2018). Spectrahedral shadows. *SIAM Journal on Applied Algebra and Geometry*, 2(1):26–44.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Schrijver, A. et al. (2003). *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Schuetz, M. J., Brubaker, J. K., and Katzgraber, H. G. (2022). Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377.

Scott, A. D. and Sokal, A. D. (2005). The repulsive lattice gas, the independent-set polynomial, and the lovász local lemma. *Journal of Statistical Physics*, 118(5):1151–1261.

Selsam, D. and Bjørner, N. (2019). Guiding high-performance sat solvers with unsat-core predictions. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 336–353. Springer.

Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. (2018). Learning a sat solver from single-bit supervision.

Seo, Y., Loukas, A., and Perraudin, N. (2019). Discriminative structural graph classification.

Shawe-Taylor, J., Cristianini, N., et al. (2004). *Kernel methods for pattern analysis*. Cambridge university press.

Shearer, J. B. (1985). On a problem of spencer. *Combinatorica*, 5:241–245.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Sinkhorn, R. (1964). A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879.

Slot, L. and Laurent, M. (2023). Sum-of-squares hierarchies for binary polynomial optimization. *Mathematical Programming*, 197(2):621–660.

Smith, K. A. (1999). Neural networks for combinatorial optimization: a review of more than a decade of research. *INFORMS Journal on Computing*, 11(1):15–34.

Spencer, J. (1977). Asymptotic lower bounds for ramsey functions. *Discrete Mathematics*, 20:69–76.

Spielman, D. A. and Teng, S.-H. (2004). Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)*, 51(3):385–463.

Sun, H., Guha, E. K., and Dai, H. (2022). Annealed training for combinatorial optimization on graphs. *arXiv preprint arXiv:2207.11542*.

Svitkina, Z. and Fleischer, L. (2011). Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM Journal on Computing*, 40(6):1715–1737.

Szegedy, M. (2013). The lovász local lemma–a survey. In *International Computer Science Symposium in Russia*, pages 1–11. Springer.

Tawarmalani, M. and Sahinidis, N. V. (2002). Convex extensions and envelopes of lower semi-continuous functions. *Mathematical Programming*, 93(2):247–263.

Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*.

Toenshoff, J., Ritzert, M., Wolf, H., and Grohe, M. (2019). Run-csp: Unsupervised learning of message passing networks for binary constraint satisfaction problems. *arXiv preprint arXiv:1909.08387*.

Toenshoff, J., Ritzert, M., Wolf, H., and Grohe, M. (2021). Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3:98.

Toth, P. and Vigo, D. (2002). *The vehicle routing problem*. SIAM.

Traud, A. L., Mucha, P. J., and Porter, M. A. (2012). Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180.

Trench, W. F. (1974). Inversion of toeplitz band matrices. *Mathematics of computation*, 28(128):1089–1095.

Valmeekam, K., Olmo, A., Sreedharan, S., and Kambhampati, S. (2022). Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*.

Van den Bout, D. E. and Miller, T. (1989). Improving the performance of the hopfield-tank neural network through normalization and annealing. *Biological cybernetics*, 62(2):129–139.

Vandenberghe, L. and Boyd, S. (2004). *Convex optimization*, volume 1. Cambridge university press Cambridge.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Veldt, N., Gleich, D. F., and Mahoney, M. W. (2016). A simple and strongly-local flow-based method for cut improvement. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1938–1947. JMLR.org.

Veličković, P., Badia, A. P., Budden, D., Pascanu, R., Banino, A., Dashevskiy, M., Hadsell, R., and Blundell, C. (2022). The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, pages 22084–22102. PMLR.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations*.

Veličković, P., Ying, R., Padovano, M., Hadsell, R., and Blundell, C. (2019). Neural execution of graph algorithms. In *International Conference on Learning Representations*.

Velickovic, P., Ying, R., Padovano, M., Hadsell, R., and Blundell, C. (2020). Neural execution of graph algorithms. In *International Conference on Learning Representations*.

Veličković, P. and Blundell, C. (2021). Neural algorithmic reasoning. *Patterns*, 2(7):100273.

Veličković, P., Ying, R., Padovano, M., Hadsell, R., and Blundell, C. (2020). Neural execution of graph algorithms. In *International Conference on Learning Representations*.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Vizel, Y., Weissenbacher, G., and Malik, S. (2015). Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035.

Vlastelica, M., Paulus, A., Musil, V., Martius, G., and Rolínek, M. (2019). Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175*.

Von Neumann, J. (1953). A certain zero-sum two-person game equivalent to the optimal assignment problem. *Contributions to the Theory of Games*, 2(0):5–12.

Vondrák, J. (2008). Optimal approximation for the submodular welfare problem in the value oracle model. In *Symposium on Theory of Computing (STOC)*.

Walteros, J. L. and Buchanan, A. (2020). Why is maximum clique often easy in practice? *Operations Research*, 68(6):1866–1895.

Wang, D., Fountoulakis, K., Henzinger, M., Mahoney, M. W., and Rao, S. (2017). Capacity releasing diffusion for speed and locality. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3598–3607. JMLR. org.

Wang, H. P., Wu, N., Yang, H., Hao, C., and Li, P. (2022). Unsupervised learning for combinatorial optimization with principled objective relaxation. In *Advances in Neural Information Processing Systems*.

Wang, P.-W., Donti, P. L., Wilder, B., and Kolter, Z. (2019). Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. *arXiv preprint arXiv:1905.12149*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Williamson, D. P. and Shmoys, D. B. (2011). *The design of approximation algorithms*. Cambridge university press.

Wu, C., Rajeswaran, A., Duan, Y., Kumar, V., Bayen, A. M., Kakade, S., Mordatch, I., and Abbeel, P. (2018). Variance reduction for policy gradient with action-dependent factorized baselines. In *International Conference on Learning Representations*.

Xu, H., Hui, K.-H., Fu, C.-W., and Zhang, H. (2020a). Tilingnn: learning to tile with self-supervised graph neural network. *ACM Transactions on Graphics (TOG)*, 39(4):129–1.

Xu, K., Boussemart, F., Hemery, F., and Lecoutre, C. (2007). Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial intelligence*, 171(8-9):514–534.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Xu, K., Li, J., Zhang, M., Du, S., Kawarabayashi, K., and Jegelka, S. (2020b). What can neural networks reason about? In *Int. Conf. on Learning Representations (ICLR)*.

Xu, K., Li, J., Zhang, M., Du, S. S., Kawarabayashi, K.-i., and Jegelka, S. (2019). What can neural networks reason about? *arXiv preprint arXiv:1905.13211*.

Xu, K. B. (2007). Benchmarks with hidden optimum solutions for graph problems. *URL http://www. nlsde. buaa. edu. cn/kexu/benchmarks/graph-benchmarks. htm*.

Yan, X., Cheng, H., Han, J., and Yu, P. S. (2008). Mining significant graph patterns by leap search. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 433–444.

Yan, Y., Swersky, K., Koutra, D., Ranganathan, P., and Hashemi, M. (2020). Neural execution engines: Learning to execute subroutines. *Advances in Neural Information Processing Systems*, 33.

Yanardag, P. and Vishwanathan, S. (2015). Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374.

Yao, W., Bandeira, A. S., and Villar, S. (2019). Experimental performance of graph neural networks on random instances of max-cut. In *Wavelets and Sparsity XVIII*, volume 11138, page 111380S. International Society for Optics and Photonics.

Yehuda, G., Gabel, M., and Schuster, A. (2020). It's not what machines can learn, it's what we cannot teach. *arXiv preprint arXiv:2002.09398*.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983.

Yolcu, E. and Poczos, B. (2019). Learning local search heuristics for boolean satisfiability. In *Advances in Neural Information Processing Systems*, pages 7990–8001.

Yu, Y., Wang, T., and Samworth, R. J. (2015). A useful variant of the davis–kahan theorem for statisticians. *Biometrika*, 102(2):315–323.

Yujia, L., Daniel, T., Marc, B., Richard, Z., et al. (2016). Gated graph sequence neural networks. In *International Conference on Learning Representations*.

Zhang, H. and Yu, T. (2020). Alphazero. *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pages 391–415.

Zheng, X., Aragam, B., Ravikumar, P. K., and Xing, E. P. (2018). Dags with no tears: Continuous optimization for structure learning. *Advances in neural information processing systems*, 31.

Nikos Karalias

nikolaos.karalias@epfl.ch
website: stalence.github.io

Education

PhD Candidate | ***Research on Machine Learning for Combinatorial and Algorithmic tasks*** 2019 – present
Supervised Pierre Vandergheynst at École polytechnique fédérale de Lausanne (EPFL)

Master's Degree in Informatics | ***Thesis: Spectral Graph Theory and Deep Learning on Graphs*** 2015 – 2017
Supervised by Anastasios Tefas at Aristotle University of Thessaloniki (AUTH)

Bachelor's Degree in Informatics | ***Thesis: Graph Spectra and Signal Processing on Graphs*** 2011 – 2015
Supervised by Ioannis Pitas at Aristotle University of Thessaloniki (AUTH)

Research

- Nikolaos Karalias, Joshua David Robinson, Andreas Loukas, Stefanie Jegelka "Neural Set Function Extensions: Learning with Discrete Functions in High Dimensions." NeurIPS 2022
- Giorgos Bouritsas, Andreas Loukas, Nikolaos Karalias, Michael M. Bronstein. "Partition and Code: learning how to compress graphs." NeurIPS 2021
- Nikolaos Karalias and Andreas Loukas. "Erdős Goes Neural: an Unsupervised Learning Framework for Combinatorial Optimization on Graphs." NeurIPS 2020 (**Oral**: 1.1% acceptance rate).
- Nikolaos Karalias, Joshua David Robinson, Andreas Loukas, Stefanie Jegelka "Neural Extensions: Training Neural Networks with Set Functions." OpenReview 2021

Research Visits

Visiting Researcher | **Stefanie Jegelka group, MIT**
Collaboration on continuous extensions of set functions

Professional Service

I have served as a reviewer for conferences and journals.

- **Conferences**: NeurIPS 2021, ICLR 2022 (Highlighted Reviewer), ICML 2022, NeurIPS 2022 (Top Reviewer), LOG 2022 (Top 20 Reviewer), ICML 2023
- **Journals**: JMLR, IEEE Transactions on Neural Networks and Learning Systems

I have also supervised student projects throughout the course of my PhD.

- Master's thesis of Shengzhao Xia. The thesis involved conducting an experimental comparison of autoregressive and one-shot policies for reinforcement learning on classic combinatorial optimization problems using Graph Neural Networks.

Teaching Experience

Teaching Assistant for courses:

- **Matrix Analysis (EPFL)**. I taught at practice sessions and I was involved in designing and grading the final exams as well as course assignments.
- **Network Tour of Data Science (EPFL)**. I designed and graded problem sets, mentored students on projects, and taught at practice sessions.

- **Information, Computation and Communication (EPFL)**. I was in charge of programming and theory sessions. I was also involved in grading projects and exams.
- **Social Media (EPFL)**. Mentored student teams on group projects. I was also involved in grading individual and group projects.
- **Signals and Systems (AUTH)** I was involved in designing and grading problem sets for exams. I also taught and supervised students at lab sessions.

Honors and Awards

Academic Performance Scholarship Fall 2016
Third semester of Master's Degree at Aristotle University of Thessaloniki

Academic Performance Scholarship Spring 2016
Second semester of Master's Degree at Aristotle University of Thessaloniki

Skills

Languages: Greek (Native), English (C2), German (B2), Bosnian
Programming: Python, Pytorch/Pytorch Geometric, C++, LaTeX, GLSL

Other Experience

- Contributed to the Armadillo open source linear algebra library.
- I enjoy studying aspects of procedural computer graphics. Shaders that I have implemented can be found at: shadertoy.com/user/Aspect