Efficient and differentiable combinatorial optimization for visual computing

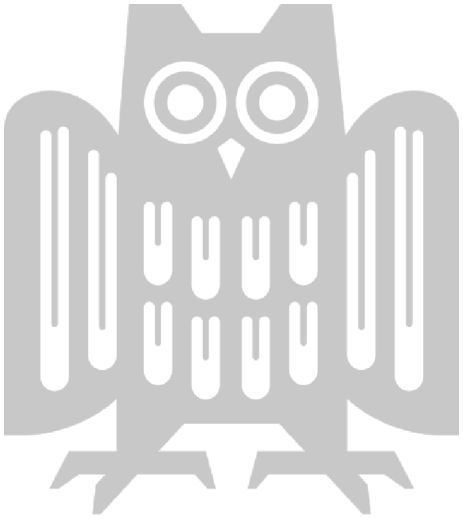
Ahmed Abbas

A dissertation submitted towards the degree

Doctor of Engineering Science (Dr.-Ing.)

of the Faculty of Mathematics and Computer Science
of Saarland University

Saarbrücken, 2024.



Date of Colloquium:	02.08.2024
Dean of the Faculty:	Prof. Dr. Roland Speicher
Chair of the Committee:	Prof. Dr. Isabel Valera
Reviewer, Advisor:	Prof. Dr. Paul Swoboda
Reviewer:	Prof. Dr. Bernt Schiele
Reviewer:	Dr. M. Pawan Kumar
Academic Assistant:	Dr. Jonas Fischer

Abstract

Many visual computing tasks involve reasoning over structured domains and discrete objects which can be modeled as combinatorial optimization (CO) problems. Tremendous speed-up in general-purpose CO solvers has allowed to solve these problems in many cases despite being NP-hard. Approaching large-scale structured prediction problems from a combinatorial optimization standpoint however, has been a challenge due to a variety of reasons. These include near real-time performance requirements and lack of differentiability of CO approaches. The latter causes difficulties in harnessing machine learning to specify problem specifications and in developing learnable CO algorithms. We aim to address these shortcomings on multiple avenues.

First, we focus on a specific CO problem for clustering known as the multicut problem with many applications in a variety of visual computing tasks. The multicut problem is defined on a weighted graph where edge weights encode preferences of the endpoints belonging to the same or different clusters. We devise methods for reducing human effort in multicut model specification, (i.e., edge weights and graph structure) and for reducing high compute times associated with large problem sizes. For inferring the edge weights, we utilize neural networks and propose strategies for improving gradient estimation through the multicut problem. This allows to train the neural network by imposing a loss on the clustering produced by the multicut problem. Our approach improves performance on large-scale panoptic segmentation tasks. As a second step, we focus on the challenge of graph structure design. We propose a compact formulation of the multicut problem on the complete graph along with efficient algorithms, sidestepping the problem of graph selection. Our approach yields better panoptic segmentation performance as compared to the one obtained by a hand-designed graph structure. Lastly, we address scalability issues of the multicut problem. We devise a massively parallelizable GPU-friendly algorithm resulting in more than an order of magnitude speed-up over existing sequential methods.

As a second avenue, we shift our focus to general-purpose CO solvers and tackle challenges related to their scalability. As a first step, we devise parallel algorithms that can harness GPUs gaining up to an order of magnitude speed-up over sequential CPU counterparts. For the second step, we exploit machine learning in solving relaxations of CO problems. We employ differentiable solver primitives and use neural networks inside the solver to guide the solving process. Given a few problem instances from a task of interest, our general-purpose solver can be adapted by learning instead of task-specific solver development. Empirically our learned approach achieves significantly better performance than its non-learned version, better solutions than task-specific solvers, and exhibits better anytime performance compared to a commercial solver (Gurobi). Our method is also applicable to some problems unrelated to structured prediction and offers competitive performance to the commercial solver.

In summary, we study methods to make CO for visual computing more practical by devising differentiable, massively parallel, and data-driven methods.

Zusammenfassung

Bei vielen Problemstellungen im Bereich Visual Computing geht es um Gebiete mit spezifischer Struktur und diskrete Objekte, die als Probleme der kombinatorischen Optimierung (CO) modelliert werden können. Die enorme Beschleunigung bei Allzweck-CO-Lösern hat es in vielen Fällen ermöglicht, diese Probleme zu lösen, obwohl sie NP-schwer sind. Das Lösen solcher großen strukturierte Vorhersageprobleme unter dem Gesichtspunkt der kombinatorischen Optimierung ist jedoch aus verschiedenen Gründen eine Herausforderung. Dazu gehören Rechenzeiten in nahezu Echtzeit und mangelnde Differenzierbarkeit von CO-Ansätzen. Letzteres führt zu Schwierigkeiten bei der Nutzung maschinellen Lernens zur Spezifizierung von Problemspezifikationen und bei der Entwicklung lernbarer CO-Algorithmen. Unser Ziel ist es, diese Mängel auf mehreren Wegen zu beheben.

Zunächst konzentrieren wir uns auf ein spezifisches CO-Problem für Clustering, das als Multicut-Problem bekannt ist, und bei vielen Anwendungen in einer Vielzahl von Problemen im Bereich Visual Computing auftritt. Das Multicut-Problem wird in einem gewichteten Graph definiert, in dem Kantengewichte die Präferenzen der Endpunkte kodieren, die zu demselben oder zu verschiedenen Clustern gehören. Wir entwickeln Methoden zur Reduzierung des menschlichen Aufwands bei der Multicut-Modellspezifikation (d.h. Kantengewichte und Graphenstruktur) und zur Reduzierung der hohen Rechenzeiten, die mit großen Problemgrößen verbunden sind. Zur Ableitung der Kantengewichte nutzen wir neuronale Netze und schlagen Strategien zur Verbesserung der Gradientenschätzung im Graphen des MulticutProblem vor. Dies ermöglicht es neuronale Netzwerk zu trainieren, indem die Clusterung, die durch das Multicut-Problem erzeugt wurde, durch eine Kostenfunktion zu quantifizieren. Unser Ansatz verbessert die Leistung bei groß angelegten panoptischen Segmentierungsaufgaben. Im zweiten Schritt konzentrieren wir uns auf die Herausforderungen des Designs der Graphenstruktur. Wir schlagen eine kompakte Formulierung des Multicut-Problems für den gesamten Graphen zusammen mit effizienten Algorithmen vor. Damit umgehen wir das Problem der Graphenauswahl. Unser Ansatz führt zu einer besseren panoptischen Segmentierungsleistung im Vergleich zu einer von Hand entworfenen Graphentruktur. Abschließend befassen wir uns mit Skalierbarkeitsproblemen des Multicut-Problems. Wir entwickeln einen massiv parallelisierbaren GPU-freundlichen Algorithmus, der zu einer Geschwindigkeitssteigerung von mehr als einer Größenordnung gegenüber bestehenden sequentiellen Methoden führt.

Als Zweites verlagern wir unseren Fokus auf universelle CO-Löser und gehen Herausforderungen im Zusammenhang mit ihrer Skalierbarkeit an. Zunächst entwickeln wir parallele Algorithmen, welche GPUs nutzen können und die gegenüber sequentiellen CPU-Algorithmen eine Geschwindigkeitssteigerung von bis zu einer Größenordnung erreichen. Im Folgenden nutzen wir maschinelles Lernen zur Lösung von Relaxierungen von CO-Problemen. Wir verwenden differenzierbare Löser-Primitive und nutzen neuronale Netze innerhalb des Löser, um den Lösungsprozess zu steuern. Anhand einiger Instanzen einer relevanten Problemstellung kann unser Allzwecklöser durch Lernen angepasst werden und somit muss kein aufgabenspezifischer Löser entwickelt werden. Empirisch erreicht unser

erlernter Ansatz eine deutlich bessere Leistung als seine nicht erlernte Version, bessere Lösungen als aufgabenspezifische Löser und weist im Vergleich zu einem kommerziellen Löser (Gurobi) in allen Testinstanzen eine bessere Leistung auf. Unsere Methode ist auch auf einige Probleme anwendbar, die nichts mit strukturierter Vorhersage zu tun haben, und bietet eine konkurrenzfähige Leistung gegenüber kommerziellen Lösern.

Im Wesentlichen untersuchen wir Methoden, um CO für Visual Computing praktischer zu machen, indem wir differenzierbare, massiv parallele und datengesteuerte Methoden entwickeln.

Acknowledgements

I express my gratitude to my advisor Paul Swoboda for his invaluable guidance during my PhD. Our meetings were always stimulating through which I learned not only about our core research areas but also about efficient & maintainable software implementations, and effective scientific writing. I am also grateful to Paul for giving enough independence to discover my own research interests while also providing helpful feedback along the way. Lastly, I admire Paul's creativeness in devising whimsical acronyms for many of our joint projects which brought great humor on looming submission deadlines.

I thank Bernt Schiele for his much needed advice at the start of my PhD. I am also grateful to Bernt for providing a great environment for research in the D2 group by providing enough research freedom, weekly seminars, biannual retreats etc. The group meetings were very beneficial through which I got exposed to topics outside of my research areas.

I would also like to thank the committee members and especially the reviewers for providing valuable feedback on the thesis.

Lastly, I thank Connie Balzert for her help and advice regarding all official matters, visa issues, etc., which made my stay very comfortable. I also thank the IT support staff of MPI, with whom I had very less chance to interact with due to the excellent IT infrastructure.

Contents

Contents 9

1 Introduction 13
1.1 Contributions and Outline . . . 13
1.1.1 Navigating NP-Hardness . . . 15

I Efficient & differentiable multicut algorithms 17

2 Background 18
2.1 Multicut Problem . . . 18
2.1.1 Integer Linear Program . . . 19
2.1.2 Related Problems . . . 20
2.1.3 Algorithms . . . 21
2.1.4 Applications . . . 23

3 Panoptic Segmentation with End-to-end Training 24
3.1 Introduction . . . 24
3.2 Related Work . . . 26
3.2.1 Panoptic Segmentation . . . 26
3.2.2 Algorithms as a Layer in Neural Networks . . . 27
3.3 Method . . . 28
3.3.1 Convolutional neural network (CNN) architecture . . . 29
3.3.2 (Asymmetric) Multiway Cut . . . 29
3.3.3 Fully Differentiable Training . . . 30
3.4 Experiments . . . 34
3.4.1 Datasets . . . 34
3.4.2 Training . . . 34
3.4.3 Results . . . 35
3.4.4 Limitations . . . 38
3.5 Conclusion . . . 39

4 Massively Parallel Multicut Algorithms 40
4.1 Introduction . . . 40
4.2 Related Work . . . 41
4.3 Method . . . 42
4.3.1 Primal: Parallel Edge Contraction . . . 42
4.3.2 Dual: Conflicted Cycles & Message Passing . . . 44
4.3.3 Primal-Dual Updates . . . 46
4.3.4 GPU Implementations . . . 48
4.4 Experiments . . . 49

4.4.1 Results . . . 51
4.5 Conclusion . . . 53

5 Efficient Multicut on Complete Graphs 54
5.1 Introduction . . . 54
5.2 Related Work . . . 55
5.3 Method . . . 56
5.3.1 Greedy Additive Edge Contraction . . . 56
5.3.2 Lazy Edge Contraction . . . 60
5.3.3 Varying Affinity Strength . . . 60
5.3.4 Computational Complexity . . . 62
5.4 Experiments . . . 63
5.4.1 ImageNet Clustering . . . 64
5.4.2 Panoptic Segmentation . . . 65
5.5 Conclusion . . . 68

II Efficient & differentiable ILP solver 69

6 Background 70
6.1 Binary Programs . . . 70
6.2 Lagrangean Decomposition . . . 72
6.3 Lagrangean Optimization . . . 74
6.3.1 Dual Block Coordinate Ascent . . . 74
6.3.2 Binary Decision Diagrams . . . 77
6.4 Common Approaches for Structured Prediction . . . 79
6.4.1 Optimization & Heuristics coupled with Neural Networks . . . 79
6.4.2 Custom Neural Architectures . . . 79

7 Massively Parallel 0–1 ILP Algorithms 81
7.1 Introduction . . . 81
7.2 Related Work . . . 82
7.3 Method . . . 84
7.3.1 Dual Optimization . . . 84
7.3.2 Primal Rounding . . . 90
7.4 Experiments . . . 91
7.4.1 Results . . . 93
7.4.2 Limitations . . . 95
7.5 Conclusion . . . 96

8 Learning to Solve 0–1 ILP Relaxations 97
8.1 Introduction . . . 97
8.2 Related Work . . . 99
8.3 Method . . . 100
8.3.1 Lagrange Decomposition . . . 100
8.3.2 Optimization of Lagrangean Dual . . . 100

8.3.3 Backpropagation through Dual Optimization . . . 101
8.3.4 Non-parametric Updates . . . 104
8.3.5 Graph Neural Network . . . 104
8.3.6 Overall Pipeline . . . 106
8.4 Experiments . . . 107
8.4.1 Results . . . 109
8.4.2 Limitations . . . 112
8.5 Conclusion . . . 112

9 Conclusion **114**

List of Algorithms **116**

List of Figures **117**

List of Tables **118**

Bibliography **119**

Introduction 1

Structured output prediction tasks are prevalent in visual computing e.g., tracking, clustering, matching, pose estimation, etc. Combinatorial optimization (CO) offers a general framework to model such problems. Such a framework is particularly attractive because of its flexibility to domain changes i.e., a method for tracking objects in driving videos will be equally applicable in different parts of the world, a formulation of clustering grid graphs can be easily adapted for meshes, etc. Since many CO problems can be formulated as integer programs, a general-purpose integer programming solver offers a potential to handle such problems conveniently

Despite the above-mentioned benefits CO approaches also come with drawbacks. These include large compute times and reduced capabilities to leverage labeled examples requiring extensive hand engineering. Tremendous progress in deep learning architectures fueled by the processing power of GPUs and the availability of training data has offered an alternative to such CO approaches for structured prediction. Many structured tasks can also be handled by specially designed neural network architectures. Such architectures however, are only applicable in specific scenarios.

In this thesis we aim to improve synergy between combinatorial optimization and deep learning based methods at multiple fronts. We study methods to address model specification issues associated with combinatorial optimization methods. Moreover we devise efficient solvers by devising massively parallelizable components and by exploiting machine learning. We study two classes of combinatorial optimization methods and split the thesis accordingly. In the upcoming Section 1.1 we summarize our contributions with an overview of each chapter.

1.1 Contributions and Outline

The thesis is divided into two parts, with each part starting with its necessary background and related works.

Part I

In this part we focus on a particular class of methods for clustering graph structured data through the multicut problem (a.k.a. correlation clustering). The multicut problem is defined on a weighted graph where edge weights describe the preference of the respective endpoints to be in the same or in different clusters. The multicut problem is particularly appealing because it does not require the number of clusters to be known. The multicut problem and its variants have been extensively utilized in computer vision for a variety of tasks including instance segmentation, multi-object tracking, pose estimation, motion segmentation, etc. We aim to make multicut model specification seamless and devise efficient parallel algorithms for its solution with lower bound guarantees.

Chapter 3: In this chapter we study a pipeline containing neural networks followed by a combinatorial optimization (CO) problem with the goal of end-to-end training. The neural network predicts the costs of the CO problem and a loss is imposed on the solution of the CO problem. A fundamental challenge in this regard is the non-differentiability of CO solutions w.r.t. the input objective. We propose a technique for improved gradient estimation through the CO problem leading to faster training convergence. As a use case we consider the problem of panoptic segmentation in images i.e., assign each pixel a category and delineate instances of each category. The panoptic segmentation problem is phrased as asymmetric multiway cut problem (Kroeger et al., 2014), a variant of the multicut problem. Our pipeline outperforms other comparable approaches in the literature due to end-to-end training. This chapter is based on our study (Abbas and Swoboda, 2021).

As a drawback, our pipeline does not offer real-time performance as solving the optimization problem takes more than a few seconds. We alleviate this issue in the following Chapter 4.

Chapter 4: In this chapter we focus on making algorithms for the multicut problem more efficient by exploiting GPU parallelism. To this end, we devise primal-dual algorithms to yield upper and lower bounds respectively. The primal step performs edge contractions on multiple regions of the multicut problem graph in parallel. The dual step optimizes a Lagrange relaxation in parallel yielding lower bounds to the objective. It also helps in obtaining better primal updates. Our algorithms outperform existing multicut algorithms by more than an order of magnitude. Such an approach, therefore, alleviates the issue of high compute times from the previous chapter. This chapter is based on the work presented as (Abbas and Swoboda, 2022b).

Chapter 5: In the last chapter of this part, we address another aspect regarding the practicality of the multicut problem for computer vision applications. The multicut problem is defined on a graph whose structure is not always trivial to design. Ideally one would like to consider the complete graph however, such an approach does not scale to large problems. This is because explicit storage of edge weights on the complete graph requires quadratic memory in the number of nodes. We alleviate this issue by a problem formulation that operates on feature vectors associated with each node. The edge costs are then computed on the fly when needed. We show how to rewrite classical greedy algorithms for multicut in our dense setting and how to modify them for greater efficiency and solution quality. Through this approach, we replace the hand-designed graph structure of Chapter 3 with our formulation yielding improved panoptic segmentation results. This chapter is based on our study in (Abbas and Swoboda, 2023).

Part II

In this second part, we study methods for an efficient and general-purpose integer programming (IP) solver as a means to address combinatorial optimization problems. Such an approach if successful, offers an alternate pathway for achieving efficient IP algorithms for structured prediction tasks. This is because a conventional approach for improving efficiency has been to devise problem-specific algorithms requiring considerable human effort each time a new problem class is encountered. First, we design a parallelizable general-purpose IP solver that can harness GPU capabilities for faster computation. Secondly, we make components of such a solver differentiable and employ neural networks

inside the solver to guide the solving process. This allows us to train our general-purpose solver on the problem class of interest in a matter of hours to a few days for further performance improvement. In our experiments, our trained solvers outperform problem-specific hand-designed solvers from the literature for structured prediction tasks. We also evaluate our approach on benchmarks outside of structured prediction and show competitive performance to a general-purpose commercial solver.

Chapter 7: In this chapter we design an IP solver containing massively parallelizable components. We build on the Lagrangean decomposition framework of Lange and Swoboda (2021) where the associated subproblems are represented by binary decision diagrams. To optimize the Lagrangean we propose a parallelizable block coordinate ascent scheme. Moreover we exploit second-order information during Lagrangean optimization through quasi-Newton updates. For recovering a primal solution we propose a parallelizable heuristic based on cost perturbation. Our algorithms yield an order of magnitude speed-up as compared to the scheme of (Lange and Swoboda, 2021) and also reach better objectives.

This chapter is primarily based on the study (Abbas and Swoboda, 2022a) and additionally from Roetzer et al. (2024). The work (Roetzer et al., 2024) has two equal main contributors and aims towards an efficient approach for integer programming based formulation for 3D shape matching. Such a pipeline allows to impose geometric consistency (i.e., neighboring elements of one shape to match to neighboring ones) in the shape matching task. One main contributor to Roetzer et al. (2024) is Paul Roetzer who proposed a better shape matching IP formulation by utilizing deep neural networks, performed experiments, and compared against approaches from the literature. The other main contribution is the thesis author who devised a second-order update scheme for optimizing the Lagrangean relaxation yielding faster convergence and also did experiments for comparing IP algorithms. In this chapter we additionally compare this second-order scheme from Roetzer et al. (2024) on the datasets considered in Abbas and Swoboda (2022a).

Chapter 8: In this chapter we aim to speed up a crucial component of IP solvers i.e., the relaxation algorithm. To this end, we generalize the Lagrangean update scheme from Chapter 7 and use graph neural networks to guide the solving process. To escape fixed points associated with block coordinate ascent algorithms we provide the neural network means to escape them. Training of our solver is done in a self-supervised fashion. Our approach achieves significantly faster performance and better objectives than its hand-designed version from Chapter 7, achieving close to optimal objectives on large structured prediction problems and other combinatorial ones. In particular, we achieve better objective values than specialized solvers for specific problem classes while retaining their efficiency. Our solver has better any-time performance over a large time period compared to a commercial solver. This chapter is mainly based on the paper (Abbas and Swoboda, 2024) and additionally includes a comparison with the second-order update scheme from Chapter 7.

1.1.1 Navigating NP-Hardness

We would like to highlight that the CO problems we study are NP-Hard. While traditional solvers such as (Gurobi Optimization, 2019; CPLEX, 2019) can often solve such problems

to guaranteed optimality, the process is frequently time-consuming. The latter phenomenon is encountered quite often on structured prediction tasks due to their large-scale nature. In many such tasks the importance of reduced computation time outweighs even that of achieving optimal solutions with guarantees. Therefore the aim of our work would mainly be on devising CO algorithms that can achieve satisfactory solutions quickly rather than striving for optimal ones. Such a design choice however, can make our algorithms less desirable in scenarios where convergence and optimality guarantees are of utmost importance. Nonetheless, in some cases our methods will provide bounds to the optimal objectives allowing to estimate the quality of our obtained solutions.

I

Efficient & Differentiable multicut algorithms

Background 2

Contents

2.1 Multicut Problem 18
2.1.1 Integer Linear Program 19
2.1.2 Related Problems 20
2.1.3 Algorithms 21
2.1.4 Applications 23

Many scientific applications require partitioning a group of objects into clusters. A common way to approach such problems is to represent the objects as a set of nodes, model their pairwise relations by edges, and use edge weights to denote pairwise similarities. The multicut problem allows partitioning of such graphs in cases the number of clusters is not known apriori. This flexibility has made the multicut problem an attractive tool in many computer vision applications. Although the multicut problem is NP-hard, heuristics do provide good enough solutions for practical purposes. Despite these favorable points, some nuisances hamper wider applicability. These include additional human effort for designing appropriate edge weights and the graph structure. Moreover multicut heuristics offer limited scalability with growing problem sizes due to their sequential nature. Our goal in this part of the thesis will be to reduce human effort in multicut model specification (edge weights, graph structure) and to design massively parallel algorithms for faster computation on GPUs. The problems of edge weights and graph structure design will be the topic of Chapters 3 and 5 respectively. Parallel algorithms for the multicut problem are proposed in Chapter 4. In the following, we provide the necessary background and discussion of related works.

2.1 Multicut Problem

A decomposition (or clustering) of a graph $G = (V, E)$ is a partitioning $\{V_i\}_{i=1}^k$ containing disjoint subsets of V where the subgraph of G induced by each V_i being connected. Each element V_i of the decomposition is termed as a cluster or a component. The set of edges straddling distinct components is a multicut of G.

Definition 1 (Multicuts)**.** *For a connected graph $G = (V, E)$ denote $\mathbb{1}_F \in \{0,1\}^E$ as the indicator vector of $F \subseteq E$ i.e., $\mathbb{1}_f = 1 \Leftrightarrow f \in F$. Moreover $G[S]$ denotes the subgraph of G induced by $S \subseteq V$. Then the set of all multicuts of G is*

$$\mathcal{M}_G = \left\{ \mathbb{1}_{\delta(V_1,\dots,V_k)} : \begin{array}{c} k \in \mathbb{N} \\ V_1 \dot\cup \dots \dot\cup V_k = V \\ G[V_i] \text{ is connected } \forall i \in [k] \end{array} \right\}. \tag{2.1}$$

where $\dot\cup$ is disjoint union, and $\delta(V_1, \dots, V_k) = \{pq \in E \mid \exists i \neq j : p \in V_i, q \in V_j\}$.

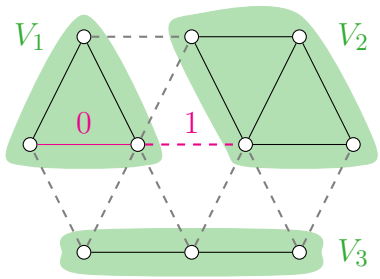


Figure 2.1: Decomposition of a graph (V, E) into three components V_1, V_2, V_3. The corresponding multicut is an edge labeling where edges of E straddling distinct components are marked by a 1 and a 0 otherwise (two of these edges labels are shown for an illustration).

See Figure 2.1 for an example decomposition of a graph into three components. Note that we assume that the graph G is connected for notational convenience. We now define the multicut problem through an additional cost vector $c \in \mathbb{R}^E$.

Definition 2 (Multicut problem)**.** *Given an graph $G = (V, E)$ and a cost vector $c \in \mathbb{R}^E$, the multicut problem is*

$$\min_{y \in \mathcal{M}_G} \sum_{ij \in E} c_{ij}\, y_{ij}. \tag{MC}$$

where $\mathcal{M}_G$ is defined in (2.1).

An edge $ij \in E$ with positive cost ($c_{ij} > 0$) favors the nodes i and j to be in the same component and is termed *attractive*. In the other case, the end points of an edge with negative cost prefer to lie in distinct components (*repulsive*).

2.1.1 Integer Linear Program

We now aim towards an integer programming formulation of the multicut problem (MC) by describing the set $\mathcal{M}_G$ (2.1) in terms of linear inequalities and binary constraints for each edge. Since the multicut problem is NP-hard (Bansal et al., 2004; Demaine et al., 2006), a complete polyhedral description of $\mathsf{conv}(\mathcal{M}_G)$ is not feasible. A good relaxation for most practical problems is given in terms of cycle inequalities (Chopra and Rao, 1993). Specifically given a cycle $C = \{e_1, \ldots, e_l\} \subseteq E$, a feasible multicut must either not contain any cut edge or should contain at least two cut edges.

Definition 1 (Cycle Inequality)**.** *For a graph $G = (V, E)$ denote cycle(G) to be a set of edges which form a cycle. A cycle inequality for a given cycle C of G is defined as*

$$\forall e \in C : y_e \leq \sum_{e' \in C \setminus \{e\}} y_{e'}. \tag{2.2}$$

By considering all possible cycles, the cycle inequality (2.2) together with the binary constraints $y \in \{0, 1\}^E$ actually define $\mathcal{M}_G$ (Chopra and Rao, 1993).

Lemma 3 (Chopra and Rao (1993)). *The feasible set of multicuts* $\mathcal{M}_G$ (2.1) *can be equivalently described as*

$$\mathcal{M}_G \equiv \{y \in \{0,1\}^E \mid y \text{ satisfies (2.2) } \forall C \in \text{cycles}(G)\}. \tag{2.3}$$

Since there can be exponentially many cycles, cutting planes can be utilized by searching for violated cycle inequalities. Moreover it is sufficient to consider simple and chordless cycles in Lemma 3 as shown by Chopra and Rao (1993). The works of Yarkony et al. (2014); Lange et al. (2018) restrict this set further to *conflicted cycles* by accounting for the structure of multicut edge costs. In general terms, a cycle is termed as conflicted if it contains exactly one repulsive edge.

Definition 2 (Conflicted cycles (Lange et al., 2018)). *Let the set of attractive edges in* E *be* $E^+ = \{c_e > 0 : \forall e \in E\}$ *and repulsive edges* $E^- = \{c_e < 0 : \forall e \in E\}$. *Then conflicted cycles of* G *is the set* $\{C \in \text{cycles}(G) : |C \cap E^-| \ = 1\}$.

Lemma 4 (Lange et al. (2018)). *Let*

$$\text{CYC}_G = \{y \in [0,1]^E \mid y \text{ satisfies (2.2) } \forall C \in \text{ cycles}(G)\}.$$

Further denote

$$\text{CYC}_G^- = \{y \in [0,1]^E \mid y \text{ satisfies (2.2) } \forall C \in \text{ cycles}(G) \text{ s.t. } |C \cap E^-| \ = 1\}.$$

Then

$$\min_{y \in \text{CYC}_G} c^\top y = \min_{y \in \text{CYC}_G^-} c^\top y. \tag{2.4}$$

To obtain a tighter relaxation further inequalities are studied in Deza et al. (1992); Chopra and Rao (1993); Oosten et al. (2001). However, for most of the problems from computer vision and machine learning the works of Lange et al. (2018); Swoboda and Andres (2017) demonstrate little to no impact by considering these additional inequalities.

2.1.2 Related Problems

There are multiple closely related problems to the multicut problem. The classical multicut problem is an extension of the min-cut problem to multiple source and sink terminals with non-negative edge costs in the work of Hu (1963). Another closely related problem is correlation clustering (Bansal et al., 2004). In this case, edges are labeled with positive or negative signs instead of real-valued costs, and the complete graph is considered. The problem was also shown to be NP-hard alongside a constant-factor approximation algorithm (Bansal et al., 2004). A weighted version of the correlation clustering problem was proposed in Demaine et al. (2006) which is equivalent (up to variable involution) to the multicut problem (MC). The authors show the equivalence of weighted correlation clustering to the classical multicut problem. An equivalent problem for multicut on complete graphs is the clique partitioning problem. The works of Grötschel and Wakabayashi (1989, 1990) study the clique partitioning problem, and propose valid inequalities and separation procedures for a cutting plane algorithm.

2.1.3 Algorithms

For obtaining solutions to (MC) without optimality guarantees or estimates on the distance to optimum, a large number of methods have been proposed with different execution time/solution quality trade-offs.

2.1.3.1 *Greedy heuristics*

A fast and simple greedy additive edge contraction (GAEC) heuristic was proposed by Keuper et al. (2015). The heuristic iteratively merges the most attractive nodes through edge contraction operations until no attractive edge is left. Its variants involving different strategies for selecting contraction candidates were proposed by Kardoost and Keuper (2018); Bailoni et al. (2022). The greedy edge fixation algorithm from Levinkov et al. (2017a) generalizes the GAEC heuristic by also marking some edges as being cut and disallowing joining moves accordingly. A comparative survey of some of the above primal heuristics was done by Levinkov et al. (2017a). Despite their approximate nature, in practice greedy algorithms perform well for computer vision and machine learning tasks as shown by the works from Keuper et al. (2015); Levinkov et al. (2017a); Bailoni et al. (2022).

In the upcoming chapters, we will build upon the greedy additive edge contraction (GAEC) heuristic of Keuper et al. (2015). The heuristic is described in Algorithm 2.1. In detail, we initialize each node as a separate cluster (all edges are cut) and iteratively contract a pair of nodes i, j having the largest non-negative cost c_{ij} (if it exists). The edge cost of parallel edges is summed up (line 3) where $\mathcal{N}_i$ corresponds to neighbors of node i. Note that contracting an edge ij corresponds to fixing $y_{ij} = 0$ in (MC).

Algorithm 2.1: `GAEC (Keuper et al., 2015)`

Data: Graph $G = (V, E)$, edge costs $c \in \mathbb{R}^E$
Result: Clusters V

```
// Check if attractive edge present
```
1 **while** $\max_{uv \in E} c_{uv} \geq 0$ **do**
2 $\quad m := ij = \arg\max_{uv \in E} c_{uv}$
$\quad$ `// Aggregate edge costs`
3 $\quad c_{ml} = c_{il} + c_{jl},\ l \in \mathcal{N}_i \cup \mathcal{N}_j \setminus \{i, j\}$
$\quad$ `// Update edges`
4 $\quad E' = \{ml \,|\, l \in \mathcal{N}_i \cup \mathcal{N}_j \setminus \{i, j\}\}$
5 $\quad E = E' \cup E \setminus \{il\}_{l \in \mathcal{N}_i} \cup \{jl\}_{l \in \mathcal{N}_j}$
$\quad$ `// Update nodes`
6 $\quad V = (V \cup \{m\}) \setminus \{i, j\}$
7 **end**

2.1.3.2 *Move-making algorithms*

The first heuristic for multicut, the classical Kernighan&Lin move-making algorithm was originally proposed by Kernighan and Lin (1970) and slightly generalized by Keuper et al. (2015). The algorithm consists of trying various moves such as joining two components, moving a node from one component to the next, etc. and performing sequences of moves that decrease the objective. A more involved Cut, Glue & Cut (CGC) move-making heuristic from Beier et al. (2014) works by alternating bipartitioning of the graph and exchanging nodes in pairs of clusters. The latter operation is performed by computing a max-cut on a planar subgraph via reduction to perfect matching. CGC was extended to a more general class of possible 'fusion moves' by Beier et al. (2016).

2.1.3.3 *Linear programming based algorithms*

For obtaining lower bounds that estimate the distance to the optimum or even certify the optimality of a solution several LP relaxation based algorithms have been proposed. These algorithms can be used inside branch and bound and their computational results can be used to guide primal heuristics to provide increasingly better solutions. It has been shown by Kappes et al. (2011); Kim et al. (2011) that multicut problems of moderate sizes can be solved with commercial integer linear programming (ILP) solvers like Gurobi (Gurobi Optimization, 2019) in a cutting plane framework in reasonable time to global optimality. A specialized algorithm for multicut on planar graphs was devised by Yarkony et al. (2012) by column generation based on solving perfect matching subproblems. Similarly Lukasik et al. (2020) employ benders decomposition allowing for parallel computation of the subproblems. The above approaches however, do not scale well to large-scale problems as the underlying LP relaxations are still solved by traditional LP solvers which do not scale linearly with problem size. Additionally, violated inequality separation (cutting planes) requires solving weighted shortest path problems which is not possible in linear time. The block coordinate ascent algorithm from Swoboda and Andres (2017) based on message passing approximately solves a dual LP relaxation of the multicut problem faster than traditional solvers. An even faster, but less powerful, approximate cycle packing algorithm was proposed by Lange et al. (2018); Lange (2020). Note that in addition to providing lower bounds, the schemes of Swoboda and Andres (2017); Lange et al. (2018) also aid in recovering better primal solutions.

2.1.3.4 *Partial optimality*

For fixing variables to their optimal values and shrinking the problem before or during optimization, persistency or partial optimality methods have been proposed in Alush and Goldberger (2012); Lange et al. (2018, 2019). These methods apply a family of criteria that, when passed, prove that any solution can be improved if its values do not coincide with the persistently fixed variables.

2.1.4 Applications

The multicut problem and its extensions such as higher-order multicut (Kim et al., 2011; Kappes et al., 2016), lifted multicut (Keuper et al., 2015), (asymmetric) multiway cut (Chopra and Rao, 1991; Kroeger et al., 2014), lifted disjoint paths (Hornakova et al., 2020) and joint multicut and node labeling (Levinkov et al., 2017c) have found numerous applications in machine learning, computer vision, biomedical image analysis, data mining and beyond. Examples include unsupervised image segmentation (Alush and Goldberger, 2013; Andres et al., 2011; Yarkony et al., 2012; Andres et al., 2013), instance-separating semantic segmentation (Kirillov et al., 2017), multiple object tracking (Tang et al., 2017; Hornakova et al., 2020), cell tracking (Jug et al., 2016), articulated human body pose estimation (Insafutdinov et al., 2017), motion segmentation (Keuper et al., 2018), image and mesh segmentation (Keuper et al., 2015), neuron segmentation for connectomics (Andres et al., 2012; Beier et al., 2017; Pape et al., 2017) and many more.

3 Panoptic Segmentation with End-to-end Training

Contents

3.1 Introduction **24**
3.2 Related Work **26**
3.2.1 Panoptic Segmentation 26
3.2.2 Algorithms as a Layer in Neural Networks 27
3.3 Method **28**
3.3.1 Convolutional neural network (CNN) architecture 29
3.3.2 (Asymmetric) Multiway Cut 29
3.3.3 Fully Differentiable Training 30
3.4 Experiments **34**
3.4.1 Datasets 34
3.4.2 Training 34
3.4.3 Results 35
3.4.4 Limitations 38
3.5 Conclusion **39**

This chapter aims for a fully differentiable architecture for simultaneous semantic and instance segmentation (a.k.a. panoptic segmentation) consisting of a convolutional neural network and an asymmetric multiway cut problem solver. The latter is a more general version of the multicut problem that elegantly incorporates semantic and boundary predictions to produce a panoptic labeling. These semantic and boundary estimates are predicted by neural networks which are then used in the objective function of the asymmetric multiway cut problem. We aim for an end-to-end differentiable pipeline where these estimates are consumed by the optimization problem in a differentiable manner. This allows to impose a loss directly on the final panoptic predictions produced by the optimization problem leading to reduced human effort and potentially better performance. For the loss function, we maximize a smooth surrogate of the panoptic quality metric. Experimental evaluation shows improvement by backpropagating through the optimization problem w.r.t. comparable approaches on Cityscapes and COCO datasets. Overall, our approach of combinatorial optimization for panoptic segmentation shows the utility of using optimization in tandem with deep learning in a challenging large-scale real-world problem and showcases benefits and insights into training such an architecture.

3.1 Introduction

Panoptic segmentation is the task of simultaneously segmenting different semantic classes and instances of the same class (Kirillov et al., 2019b). Panoptic segmentation is challenging

since neural networks (NN) may produce conflicting predictions (i.e., boundaries separating instances that are not closed contours, instance voting schemes with multiple maxima per instance, etc.). Therefore most approaches combine NNs with a post-processing step to compute a final panoptic segmentation that resolves the conflicting evidence produced by NNs. In general, joint training of NNs with post-processing algorithms is an active research area. In our work we propose a fully differentiable approach for panoptic segmentation, our post-processing being a combinatorial optimization problem.

In this work we pursue the bottom-up approach building segmentations directly from pixels and combine CNNs with the asymmetric multiway cut problem (AMWC) (Kroeger et al., 2014). The latter is an elegant combinatorial optimization problem that combines semantic and affinity predictions and directly produces a panoptic labeling. We train CNN and AMWC jointly so that the supervisory signal for training the CNN is influenced by the computations of the combinatorial optimization stage. The loss we propose to use for this training differs from common lower-level CNN losses and is a smooth surrogate closely corresponding to the final panoptic quality metric (Kirillov et al., 2019b). We show in this work how our conceptual contributions i.e., using AMWC as a differentiable module and training on surrogate panoptic quality loss can be made to work together and yield performance improvements.

The general idea of combining optimization and neural networks and train them jointly has recently enjoyed resurgent interest. The fundamental problem for the specific task of combinatorial optimization is that the output of combinatorial problem is 0–1 valued, hence the loss landscape becomes piecewise constant and simply differentiating through a solver is not possible anymore. Several methods have been proposed to address this problem (Vlastelica et al., 2019; Domke, 2010; Peng et al., 2018; Ferber et al., 2020; Berthet et al., 2020; Indelman and Hazan, 2020). To our knowledge our work is the first to utilize the perturbation techniques (Vlastelica et al., 2019; Domke, 2010) on a large-scale setting with scalable but suboptimal heuristic solvers. We give evidence that training works in this setting and gives performance benefits. To this end, we propose a robust extension of the backpropagation technique (Vlastelica et al., 2019) that gives better empirical convergence.

Our architecture is inspired by Cheng et al. (2020); Chen et al. (2018) and consists of a ResNet-50 backbone, a semantic segmentation branch for computing class costs and an affinity branch for boundary predictions. Semantic and affinity costs are taken as input by the AMWC solver that returns a panoptic labeling. We first pre-train semantic and affinity branches with simple cross-entropy losses obtaining a strong baseline that achieves a performance similar or better than other bottom-up approaches (Cheng et al., 2020; Wolf et al., 2020; Gao et al., 2019). We finetune subsequently with the AMWC solver and the panoptic surrogate loss via our new robust backpropagation approach and show further performance improvements.

Current state-of-the-art approaches use very large networks (e.g., Max-DeepLab (Wang et al., 2020a) uses transformers containing more parameters than a ResNet-101). This might lead to the impression that advances in panoptic segmentation require deeper and more sophisticated architecture. We show that our simpler model can be significantly improved by a fully differentiable approach and argue that simpler models have not yet reached their full potential. Also, our simpler architecture allows for a more controlled

setting and makes it easier to identify crucial components and measure to which extent performance improvements can be achieved.

3.2 Related Work

3.2.1 Panoptic Segmentation

We categorize panoptic segmentation approaches into three categories: (i) bottom-up methods predict information on the pixel-level and then use post-processing to produce a segmentation, (ii) top-down methods proceed by first identifying regions of interest (ROI) and subsequently basing segmentation on them and (iii) hybrid methods combine bottom-up and top-down ideas. For a general overview of recent segmentation methods we refer to Minaee et al. (2021). Here we will restrict to panoptic segmentation tasks.

Top-down. Recent works include (Li et al., 2018a; Kirillov et al., 2019b; Porzi et al., 2019; Kirillov et al., 2019a; Xiong et al., 2019; Qiao et al., 2020; Carion et al., 2020; Yang et al., 2020a; Mohan and Valada, 2021). This principle has also been used with weak supervision (Li et al., 2018b). As a drawback, top-down approaches use ROIs which are mostly axis-aligned and so they can be in-efficient for scenarios containing deformable objects (Tian et al., 2020).

Bottom-up. Panoptic-DeepLab (Cheng et al., 2020) based on (Yang et al., 2019) proposes a single-stage neural network architecture which combines instance center of mass scores with semantic segmentation to compute panoptic segmentation. They use post-processing similar to Hough-voting (Ballard, 1981), obtaining great results and reducing the gap to top-down approaches. Subsequently, Axial-DeepLab (Wang et al., 2020b) made improvements using an attention mechanism to enlarge the receptive field using the post-processing scheme of (Yang et al., 2019).

The methods SSAP (Gao et al., 2019) and SMW (Wolf et al., 2020) are most similar to our as they also use semantic and affinity scores with a graph partitioning algorithm. SMW (Wolf et al., 2020) additionally uses Mask-RCNN (He et al., 2017) and SSAP solves multiple graph partitioning problems in coarse-to-fine manner. Older works (Kirillov et al., 2017; Liu et al., 2018) use graph partitioning schemes but only for the instance segmentation task.

Hybrid. The approaches (Li et al., 2020; Wolf et al., 2020) use both bottom-up (affinity scores) and top-down (bounding boxes) sources of information. Conditional convolution (Tian et al., 2020) was used by Wang et al. (2020a). Transformers are used by Carion et al. (2020) and combined with Max-DeepLab in a sophisticated architecture, achieving remarkable results. They used a surrogate for the panoptic quality metric along with an instance discrimination loss similar to Wu et al. (2018). However, Max-DeepLab imposes an upper bound on the maximum number of instances in an image and requires thresholding low confidence predictions.

In summary, bottom-up methods are generally simpler than top-down ones and require fewer hyper-parameters. However, they lack global context and are generally outperformed by top-down approaches. As a solution Axial-DeepLab (Wang et al., 2020b) reduce this

gap by incorporating long range context.

Almost all of the above-mentioned approaches use multiple loss functions (see (Kendall et al., 2018) for a possible solution), need thresholds for getting rid of low confidence predictions or assume an upper bound on the number of instances and therefore require hyperparameter tuning. To achieve end-to-end training, approaches of Wang et al. (2020a); Carion et al. (2020); Li et al. (2020) design mechanisms embedded in the NNs which can compute panoptic segmentations directly but still have test-time hyperparameters (such as maximum number of instances, probability thresholding) and need more complicated architectures. Except for the above works, other approaches delegate this task to a post-processing module which does not participate in training. The motivation of our work is based on prioritizing ease-of-use and simplicity. Therefore we have chosen a bottom-up approach and propose a fully differentiable method for training with only one loss and no ad-hoc downstream refinements of the segmentation.

3.2.2 Algorithms as a Layer in Neural Networks

Recently there has been some interest in training neural networks with additional layers for problem-specific constraints and prior knowledge. The works of Gould et al. (2019); Kotary et al. (2021) provide an extensive survey and insights. An excellent overview of multiple approaches for learning graphical model parameters is given in Domke (2013). Since the focus of our work is on using an optimization problem as a layer in neural networks, hence we will review approaches for this scenario. The approaches can be categorized as follows:

Unrolling. For training NNs together with cheap and differentiable iterative algorithms (or for algorithms that can be made differentiable e.g., by smoothing), straightforwardly computing gradients is the most simple approach. This has been done for K-means (Wilder et al., 2019) bipartite matching (Zeng et al., 2019), conditional random fields (Zheng et al., 2015; Arnab et al., 2018; Song et al., 2019b; Domke, 2010), non-linear diffusion for image restoration (Chen and Pock, 2016) and ranking and sorting (Cuturi et al., 2019). The interesting study (Christianson, 1994) shows that under some stability conditions backpropagation through the last few steps of iterative procedures is enough to get good estimates of gradients.

Implicit function theorem. In case solutions satisfy fixed point conditions (e.g., KKT conditions) the implicit function theorem can be used to compute gradients. This was done for quadratic programs in (Amos and Kolter, 2017), embedding MaxSAT in neural networks (Wang et al., 2019), a large class of convex optimization problems (Agrawal et al., 2020), smoothed top-k selection via optimal transport (Xie et al., 2020) and deep equilibrium models (Bai et al., 2019).

Problem-specific methods. Specialized approaches for backpropagating for specific problems were investigated for submodularity (Djolonga and Krause, 2017) (e.g., using a graph-cut layer), belief propagation (Knobelreiter et al., 2020), dynamic programming (Mensch and Blondel, 2018), markov random fields (Chen et al., 2015; Kirillov et al., 2016) and nearest neighbor selection (Plötz and Roth, 2018).

Perturbation approaches. Perturbing the objective of an optimization problem for learning has been proposed by Papandreou and Yuille (2011); Li et al. (2013); Bertasius

et al. (2017) for graphical model parameters. In the works of Corro and Titov (2019); Berthet et al. (2020); Paulus et al. (2020) perturbation is used in the forward pass to get a differentiable estimate of the solution. Perturbing the objective in the direction of loss decrease has been proposed by Domke (2010) for backpropagating through graphical model inference, in McAllester et al. (2010) to estimate gradients through a structured loss and in Vlastelica et al. (2019) to backpropagate through combinatorial optimization problems. The latter was used for ranking (Rolínek et al., 2020a) and graph matching (Rolínek et al., 2020b).

3.3 Method

Our architecture shown in Figure 3.1 is comprised of two stages: (i) a CNN to compute semantic class and affinities for boundary predictions followed by (ii) an AMWC optimization layer producing the final panoptic labeling. We describe below our CNN architecture, the AMWC problem and finally the approach for backpropagting through the AMWC solver to optimize panoptic surrogate loss.

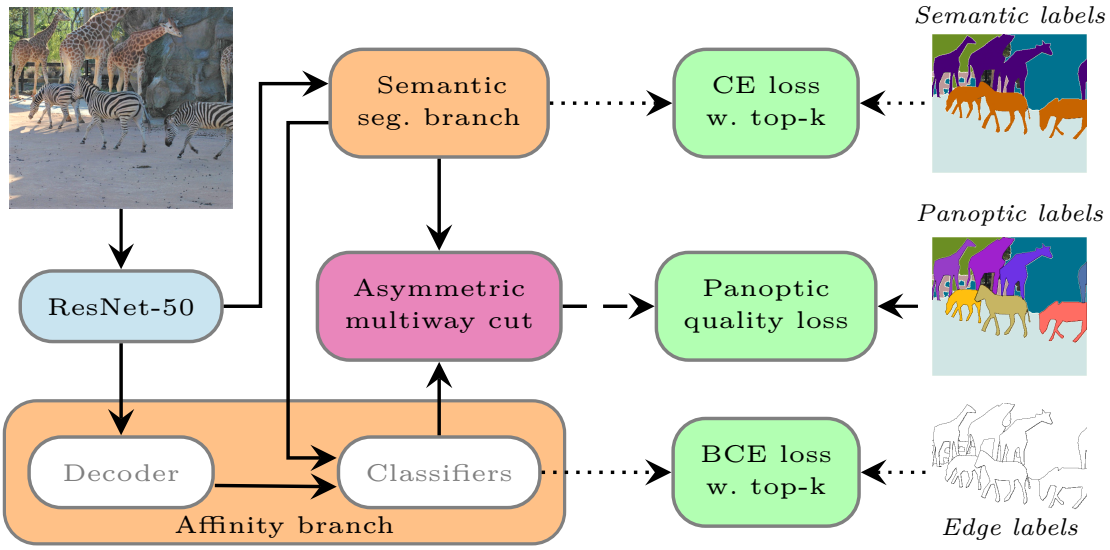


Figure 3.1: Overview of our architecture: Image features computed through a ResNet-50 backbone (He et al., 2016b) are fed into a semantic segmentation branch to predict class scores and to an affinity branch to predict object boundaries. Costs from both branches are used in the AMWC solver for computing a panoptic labeling. Pre-training of the semantic and affinity branch is done with top-k cross-entropy losses (Yang et al., 2019) (dotted arrows). For backpropagation through AMWC solver we use the panoptic quality loss (dashed arrows). The computation flow marked by solid lines is for panoptic segmentation, dotted arrow for pre-training and dashed arrows for fully differentiable training.

3.3.1 Convolutional neural network (CNN) architecture

Our CNN architecture (see Figure 3.1) is comprised of the following parts: a shared ResNet-50 backbone pre-trained on ImageNet (Deng et al., 2009) producing feature maps for the subsequent semantic and affinity branch. Our CNN architecture corresponds to Panoptic-Deeplab (Cheng et al., 2020) with the exception of a modified instance segmentation branch due to different post-processing (Hough voting for vs. AMWC in our work). We also use DeepLabv3+ (Chen et al., 2018) decoders for both semantic and affinity branch similar to Cheng et al. (2020) allowing for a fair comparison.

Affinity predictor. The affinity branch predicts for given pairs of pixels whether they belong to the same instance. It takes two sources of inputs: (i) features from the affinity decoder and (ii) semantic segmentation costs which makes finding boundaries between different classes easier. Gradients of segmentation costs computed from affinity predictors are not backpropagated during training to preclude the affinity branch from influencing the semantic branch.

We take horizontal and vertical edges at varying distances d. For COCO we use $d \in \{1, 4, 16, 32, 64\}$ and for Cityscapes $d \in \{1, 4, 16, 32, 64, 128\}$. For each d all corresponding edges are sampled and affinity scores are computed by a dedicated predictor for each distance. For long range edges with $d > 1$ we compute edge features by taking the difference of affinity features of the edge endpoints before sending them to the predictor. This helps in capturing long-range context.

3.3.2 (Asymmetric) Multiway Cut

Multiway cut (MWC) (Calinescu, 2008) is a combinatorial optimization problem for graph partitioning defined on a graph. In MWC a pre-defined number of classes is given and each node is assigned to one. The cost of a class assignment is given by node and edge affinity costs that give the preference of a node belonging to a certain class and endpoints of the edge to belong to the same class respectively. Hence, the multiway cut can be straightforwardly used to formulate semantic segmentation, each MWC class corresponding to a semantic class.

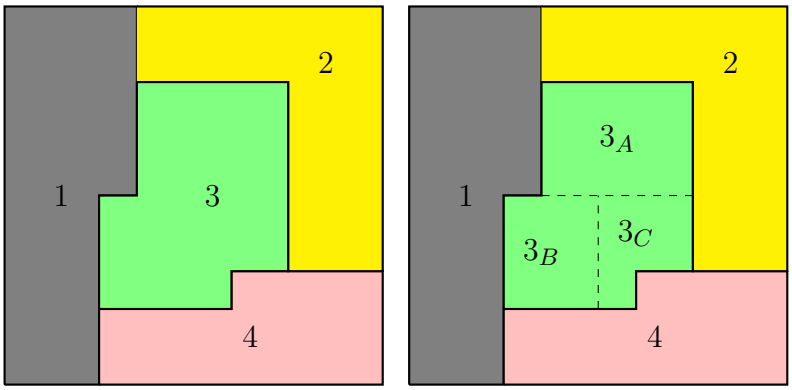
The asymmetric multiway cut (AMWC) problem was introduced by Kroeger et al. (2014) as an extension of MWC. It additionally allows to subdivide some classes into an arbitrary number of sub-clusters. This allows to model segmenting a given semantic class into multiple instances for panoptic segmentation.

Mathematically, MWC and AMWC are defined on a graph $G = (V, E)$ together with edge weights $c_E : E \to \mathbb{R}$ and node costs $c_V : V \times \{1, \dots, K\} \to \mathbb{R}$, where K is the number of classes. The edge affinities c_E indicate the preference of edge endpoints to belong to the same cluster, while the node costs c_V indicate the preference of assigning nodes to classes. A set $P \subseteq [K]$ contains classes that can be partitioned. For MWC we have $P = \varnothing$ while for AMWC $P \subseteq [K]$. Let $\mathcal{M}_G$ be the set of valid boundaries i.e., edge indicator vectors of partitions of V as defined in (2.1). The MWC and AMWC

optimization problems can be written as

$$\begin{aligned} \min_{x:V\to\{1,\dots,K\},y\in\mathcal{M}_G} \quad & \textstyle\sum_{i\in V} c_V(i,x(i)) + \sum_{ij\in E} c_E(ij)\cdot y(ij) \\ \text{s.t.} \quad & y(ij) = 0, \text{ if } x(i) = x(j) \notin P \\ & y(ij) = 1, \text{ if } x(i) \neq x(j) \end{aligned} \tag{3.1}$$

The above constraints stipulate that y produces a valid clustering of the graph compatible with the node labeling x i.e., boundaries implied by y align with class boundaries defined by x and non-partitionable classes not in P do not possess internal boundaries. The AMWC can be thought of as a special case of InstanceCut (Kirillov et al., 2017) that has class-dependent edge affinities, which however, makes it less scalable. Illustrations of MWC and AMWC are given in Figure 3.2.



(a) MWC (b) AMWC

Figure 3.2: Exemplary MWC and AMWC problems with 4 classes ($K = 4$). MWC is a special case of AMWC when $P = \varnothing$. For $P = \{3\}$ we get an AMWC problem where class 3 is partitioned into subclusters (instances) 3_A, 3_B and 3_C.

Given a feasible solution (x, y) satisfying the constraints in (3.1), the panoptic labeling $z : V \to \{1, \dots, J\}$ is computed by connected components w.r.t. y i.e., $z(i) = z(j) \Leftrightarrow y(ij) = 0,\ \forall ij \in E$. For optimizing AMWC (3.1) we adapt the greedy additive edge contraction Algorithm 2.1 (Keuper et al., 2015) to account for node costs c_V.

Note that, contrary to other approaches for panoptic segmentation such as (Wang et al., 2020a; Tian et al., 2020; Xiong et al., 2019) AMWC neither has an upper bound on the number of instances (which is automatically decided by the optimization problem) nor suffers from computational bottlenecks in this regard. It also does not require thresholding to get rid of low confidence predictions.

3.3.3 Fully Differentiable Training

To train our architecture along with the AMWC solver we first introduce a new robust variant of the perturbation technique for backpropagation (Vlastelica et al., 2019) which works well for our setting of a large-scale problem and suboptimal solver. Second, we introduce a smooth panoptic loss surrogate. Last, we show how to backpropagate gradients for the panoptic loss surrogate through a MWC layer.

3.3.3.1 *Robust Perturbation for Backpropagation*

The fundamental difficulty of backpropagating through a combinatorial optimization problem is that the loss landscape is piecewise constant, since the output of the combinatorial problem is integer valued. To handle this difficulty, generally applicable perturbation techniques (Bengio et al., 2013; Domke, 2010; Indelman and Hazan, 2020; McAllester et al., 2010; Vlastelica et al., 2019) have been proposed. They work by taking finite differences of solutions with perturbations of the original problem. The work (Vlastelica et al., 2019) interprets this as creating a continuous interpolation of the non-continuous original loss landscape.

The second difficulty is that, due to large size and NP-hardness of AMWC, we use a heuristic suboptimal solver that does not in general deliver optimal solutions. Therefore, we propose a multi-scale extension of Vlastelica et al. (2019) for increased robustness that works well in our setting.

Assume a binary integer linear optimization layer $\mathcal{W}$ takes a cost vector c as input from a neural network i.e., $\mathcal{W} : \mathbb{R}^n \to \{0,1\}^n, c \mapsto \arg\min_{x \in \mathcal{S}} \langle c, x \rangle$ where $\mathcal{S} \subset \{0,1\}^n$ is the set of constraints. Afterwards the minimizer of $\mathcal{W}$ is fed into a loss function $L : \{0,1\}^n \to \mathbb{R}$. For backpropagation we need to compute the gradient $\frac{\partial (L \circ \mathcal{W})}{\partial c}$, where $L \circ \mathcal{W}$ is the composition of L and $\mathcal{W}$. Since, this gradient is zero almost everywhere a continuous interpolation $(L \circ \mathcal{W})_\lambda$ is proposed by Vlastelica et al. (2019) where $\lambda > 0$ is an interpolation range. The gradient w.r.t. the interpolation is computed by perturbation of the cost vector c by incoming gradient as follows

$$\frac{\partial (L \circ \mathcal{W})_\lambda}{\partial c} = \frac{1}{\lambda} \Big[\mathcal{W}(c + \lambda \nabla L(\mathcal{W}(c))) - \mathcal{W}(c) \Big] \tag{3.2}$$

while Vlastelica et al. (2019) report that a large interval of interpolation ranges λ work well on their test problems with optimal solvers, we have not been able to confirm this for our suboptimal heuristic that only gives approximately good solutions to $\mathcal{W}$. Therefore, we propose to use a multi-scale loss and its gradient

$$(L \circ \mathcal{W})_{avg} := \frac{1}{N} \sum_{i=1}^{N} (L \circ \mathcal{W})_{\lambda_i}, \qquad \frac{\partial (L \circ \mathcal{W})_{avg}}{\partial c} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial (L \circ \mathcal{W})_{\lambda_i}}{\partial c} \tag{3.3}$$

where λ_i are sampled uniformly in an interval. While the robust backpropagation formula (3.3) needs multiple calls to the optimization oracle $\mathcal{W}$, they can be computed in parallel. In practice the computation time for a backward pass will hence not increase.

3.3.3.2 *Panoptic Quality Surrogate Loss*

Panoptic quality (PQ) (Kirillov et al., 2019b) is a size-invariant evaluation metric defined between a set of predicted masks and ground-truth masks for each semantic class $l \in [K]$. For each class, it requires to match predicted and object masks to each other w.r.t. intersection-over-union (IoU) since instance labels are permutation invariant. A pair of predicted and ground truth binary masks p and g of the same class l is matched (i.e., true-positive) if $IoU(p, g) \geq 0.5$. We write $(p, g) \in TP_l$. For the unmatched masks, each prediction (ground-truth) is marked as false positive FP_l (false negative FN_l). Since at

most one match exists per ground truth mask, this matching process is well-defined (Kirillov et al., 2019b). The PQ metric is defined as the mean of class specific PQ scores

$$PQ_l = \frac{\sum_{(p,g)\in TP_l} IoU(p,g)}{|TP_l| + 0.5(|FP_l| + |FN_l|)} \tag{3.4}$$

Note that the PQ score (3.4) can be arbitrarily low just by the presence of small sized false predictions (Cheng et al., 2020; Xiong et al., 2019; Porzi et al., 2019). A common practice to avoid such issue is to reject small predictions before computing the PQ score with some dataset specific size thresholds, before evaluation. However, this rejection mechanism is not incorporated during training.

The PQ metric (3.4) cannot be straightforwardly used for training due to the discontinuity of the hard threshold based matching and the rejection mechanism. Therefore we replace the hard threshold matching process for each class l by computing correspondences via a maximum weighted bipartite matching with IoU as weights. The corresponding matches are $\overline{TP}_l$, the unmatched prediction masks $\overline{FP}_l$ and the unmatched ground truth masks $\overline{FN}_l$. The hard thresholding is smoothed via soft thresholding function $h(u) = \frac{u^4}{u^4+(1-u)^4}$ centered around 0.5. The small prediction rejection mechanism for mask p is smoothed via $\sigma_l(p) = [1 + \exp(-0.1(1^T p - t_l))]^{-1}$ centered at area threshold t_l for class l. The overall surrogate PQ for class l is

$$\overline{PQ}_l = \frac{\sum_{(p,g)\in\overline{TP}_l} h(IoU(p,g))\,\sigma_l(p)\,IoU(p,g)}{\sum_{(p,g)\in\overline{TP}_l} h(IoU(p,g))\,\sigma_l(p) + 0.5\{\sum_{p\in\overline{FP}_l}\sigma_l(p) + |\overline{FN}_l|\}} \tag{3.5}$$

where the term $h(IoU(p,g))$ models the probability of a predicted mask p being true positive.

3.3.3.3 *Transformation to Multiway Cut*

In order to directly train with the panoptic loss surrogate (3.5) via the backpropagation formula (3.3) we propose a transformation of the AMWC problem to a lifted MWC problem in the backward pass for computing gradients. The AMWC optimization oracle $\mathcal{W}$ can be written as

$$\begin{aligned}(x^*, y^*, z^*) = \arg\min_{x,y,z} &\langle c_V, x\rangle + \langle c_E, y\rangle \\ \text{s.t.} \quad z(i) &= z(j), \text{ if } y(ij) = 0 \\ z(i) &\neq z(j), \text{ if } y(ij) = 1 \\ (x,y) \in \mathcal{S}, z &\in \mathbb{Z}_+\end{aligned} \tag{3.6}$$

where $\mathcal{S}$ describes the constraint listed in (3.1) and the loss is calculated w.r.t. panoptic labels z^* i.e., $\mathcal{W}(c_V, c_E) = z^*$. To compute the gradients as per (3.2) we need to perturb the cost vector associated with z in (3.6). However, AMWC only takes semantic costs and affinity costs as input not the panoptic costs. In other words, the gradient of (3.5) affects node costs of individual instances separately (i.e., they work on panoptic labels), but AMWC assumes node costs are equal for all instances of one semantic class (i.e., it works

on class labels). Therefore we transform the AMWC problem into a lifted MWC problem that has a class for each panoptic label in the ground truth. This allows to optimize directly in panoptic label space and compute a gradient w.r.t. semantic and affinity costs which can then be backpropagated to corresponding branches.

Algorithm 3.1: BACKWARD PASS

Input : $\frac{\partial L}{\partial z}$, c_V, c_E, x, y, m, λ
Output : $\frac{\partial L}{\partial c_V}$, $\frac{\partial L}{\partial c_E}$

// Transform node costs to panoptic costs and perturb
$c'_V(l) = c_V(m(l)) + \lambda \frac{\partial L}{\partial z}(l), \ \forall l \in [J]$
// Multiway cut on panoptic label space
$(z_p, y_p) = \mathrm{MWC}(c'_V, c_E)$
// Perturbed class labels
$x_p(i) = m(z_p(i)), \forall i \in V$
// Compute node cost gradients
$\frac{\partial L}{\partial c_V} = \frac{1}{\lambda}(x_p - x)$
// Compute edge cost gradients
$\frac{\partial L}{\partial c_E} = \frac{1}{\lambda}(y_p - y)$
return $\frac{\partial L}{\partial c_V}, \frac{\partial L}{\partial c_E}$

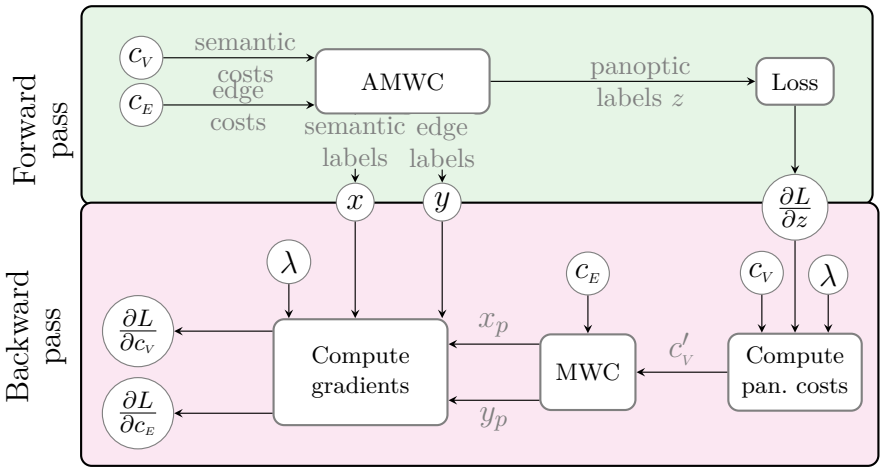


Figure 3.3: Gradient computation for c_V, c_E for fully differentiable learning: AMWC produces semantic, edge, panoptic labels x, y, z resp. Perturbations of panoptic label costs c'_V are computed and sent to the MWC solver together with the original edge costs c_E. Results are used to compute and return the gradients.

For the backward pass described in Algorithm 3.1 we define the following notation: Let J be the number of classes for the lifted MWC problem and $m : [J] \to [K]$ the mapping from panoptic labels onto the corresponding semantic class. Algorithm 3.1 computes the gradient w.r.t. the simple backpropagation formula (3.2). For the robust backprop (3.3) the algorithm has to be called multiple times with the corresponding interpolation ranges λ. An illustration of the gradient computation is given in Figure 3.3.

Line 1 in Alg. 3.1 merges two sources of information i.e., preference of the loss L on panoptic labels z and current class costs c_V. Note that the edge costs c_E are not perturbed. Afterwards, the perturbed panoptic labels z_p are converted back to class labels x_p on line 3 to compute the gradients.

3.4 Experiments

All baselines are trained on NVIDIA Quadro RTX 8000 GPUs with $48GB$ memory each. For fully differentiable training we use one Tesla P40 with $24GB$ memory and a 32 core CPU to solve all AMWC problems in the batch in parallel.

3.4.1 Datasets

We train and evaluate our approach on the Cityscapes (Cordts et al., 2016) and COCO (Lin et al., 2G014) panoptic segmentation datasets. We test on the validation and test sets provided by the two datasets. For evaluation on the test set we do not use validation set for training.

Cityscapes. Contains traffic related images of resolution 1024×2048 where training, validation and testing splits have 2975, 500, and 1525 images for training, validation, and testing, respectively. It contains 8 'thing' and 11 'stuff' classes. During training we use random scale augmentation and crop to 512×1024 resolution as done in Panoptic-DeepLab (Cheng et al., 2020). During evaluation the input images are sent at original resolution. The values of small segment rejection thresholds (used during both training and inference) are $200, 2048$ for 'thing'and 'stuff'class resp. Lastly, to handle larger occlusions we additionally use affinities at a distance of 128.

COCO. Is more diverse and contains $118k$, $5k$, and $20k$ images for training, validation, and testing, resp. The dataset has 80 'thing' and 53 'stuff' classes. During training random scale augmentation is also used with a crop size of 640×640 resolution same as in Cheng et al. (2020). The values of small segment rejection thresholds (used during both training and inference) are $200, 4096$ for 'thing'and 'stuff'class resp. During evaluation the input images are resized to 640×640 resolution.

3.4.2 Training

We closely follow the implementation of Panoptic-DeepLab (Wu et al., 2019) (based on Pytorch (Paszke et al., 2019)), use the provided ImageNet pre-trained ResNet-50 backbone and the same learning rate parameters for training our baseline model. The Adam optimizer (Kingma and Ba, 2014) is used for all our experiments.

Resolution. The CNNs produce an output with $1/4$-th the resolution in every dimension w.r.t. input images, similar to Panoptic-DeepLab. This reduced input size is maintained for AMWC (instead of upsampled) to reduce computation time during full training and evaluation. The panoptic labels computed by the AMWC solver are upsampled during evaluation. Since these labels are discrete, upsampling may misalign object boundaries

and small ground-truth objects can potentially be missed as well. While this can put our method at a disadvantage, our full training scheme offsets this by achieving panoptic quality even better than the performance at finest resolution of comparable methods.

Baseline pre-training. We pre-train the CNN architecture as a baseline model and for achieving a good initialization for the subsequent fully differentiable training. This also allows us to measure the additional gain by full training. In pre-training we apply the weighted top-k cross-entropy loss (Yang et al., 2019) to each affinity predictor separately and also to the semantic segmentation branch. Since the main objective of the affinity classifier should be to predict instance boundaries we increase the loss by a factor of 4 for edges where at least one endpoint belongs to a 'thing' class. Additionally, we also increase the semantic and affinity loss weights of small objects by a factor of 3 (Cheng et al., 2020).

We train Cityscapes on one GPU with batch-size 12 for $250k$ iterations, with initial learning rate 0.001 and the decay strategies of Panoptic-DeepLab. Training takes around 8 days. COCO is trained on four GPUs with a total batch-size of 48 for $240k$ iterations using the same learning rate parameters as above. Training takes around 11 days.

Full training. For training our pipeline through AMWC solver we use only the panoptic quality surrogate loss (3.5) and fine-tune the semantic and affinity classifiers along with the last layer of each semantic and affinity decoder. The ResNet50 backbone and all batch normalization parameters (Ioffe and Szegedy, 2015) are frozen. We train with batch size of 24 until training loss convergences which amounts to 3000 iterations for Cityscapes and 10000 iterations for COCO. To approximate the gradient (3.3) we use relatively large values of λ compared to (Vlastelica et al., 2019) since in-exact optimization might not react to small perturbations correctly (for example the backward pass solution might not even be equal to the one from the forward pass for $\lambda \to 0$). We also observed more stable training curves for larger values of N and use $N = 5$ in our experiments.

3.4.3 Results

We compare panoptic quality (in terms of percentage) on both testing PQ^{test} and validation PQ^{val} splits of Cityscapes and COCO datasets, see Table 3.1. For the testing splits evaluation requires submission to an online server. We also show performance on 'thing' classes PQ_{th}, and stuff classes PQ_{st} separately. To allow a fair comparison, we restrict ourselves to results of competing approaches which are closest to our setting i.e., without test-time augmentation, similar number of parameters in the network, not utilizing other sources of training data, etc. For an overall comparison, we also consider at least one state-of-the-art work from each other type of method (top-down, hybrid, etc.).

First, our fully trained model improves by more than 3 and 4 points in panoptic quality for Cityscapes and COCO resp. in comparison to our baseline model. This is evidence our panoptic loss surrogate and training in conjunction with the combinatorial solver works. Especially, performance on the 'thing' classes improves which have internal boundaries. We argue this is mainly due to better training of the affinity branch, which benefits more from the AMWC supervisory signal. The methods SSAP (Gao et al., 2019), SMW (Wolf et al., 2020) are closest to ours in-terms of the post-processing, and Panoptic-DeepLab in-terms of architecture resp. Our fully trained model outperforms SSAP even in a setting

Table 3.1: Results on Cityscapes (above) and COCO (below) on validation and testing splits. We divide the methods into two groups where lower half for each dataset contains the approaches which are comparable to ours with bold numbers representing the best performance in this category. R-X: ResNet-X, X-71: Xception-71, †: Mask selection (e.g., by Mask-RCNN), *: Uses test-time augmentation. (-) Marks the results which are not reported for that setting.

Method	Backbone	PQ^{test}	PQ^{test}_{th}	PQ^{test}_{st}	PQ^{val}	PQ^{val}_{th}	PQ^{val}_{st}
			Cityscapes				
Axial-DL (Wang et al., 2020b)	Axial-L	62.7	53.4	69.5	63.9	-	-
Pan-DL (Cheng et al., 2020)	X-71	60.7	-	-	63.0	-	-
Li et al. (2020)†	R-50	61.0	52.7	67.1	61.4	54.7	66.3
Xiong et al. (2019)†	R-50	-	-	-	59.3	54.6	62.7
Kirillov et al. (2019a)†	R-101	-	-	-	58.1	52.0	62.5
SSAP (Gao et al., 2019)*	R-101	58.9	48.4	**66.5**	61.1	55.0	-
Pan-DL (Cheng et al., 2020)	R-50	58.0	-	-	60.3	51.1	66.9
SMW (Wolf et al., 2020)†	Multiple	-	-	-	59.3	50.6	65.7
Li et al. (2020)	R-50	-	-	-	59.0	50.2	65.3
SSAP (Gao et al., 2019)	R-50	-	-	-	56.6	49.2	-
Our baseline	R-50	56.7	46.0	64.5	58.5	48.3	66.0
Our full	R-50	**60.0**	**51.8**	65.9	**62.1**	**55.1**	**67.2**
			COCO				
Max-DL (Wang et al., 2020a)	MaX-S	49	54	41.6	-	-	-
Li et al. (2020)†	R-50	43.6	48.9	35.6	43.4	48.6	35.5
Xiong et al. (2019)†	R-50	-	-	-	42.5	48.5	33.4
Axial-DL (Wang et al., 2020b)	Axial-S	42.2	46.5	35.7	41.8	46.1	35.2
Kirillov et al. (2019a)†	R-101	40.9	48.3	29.7	40.3	47.5	29.5
Pan-DL (Cheng et al., 2020)	X-71	38.8	-	-	39.7	43.9	33.2
SSAP (Gao et al., 2019)*	R-101	36.9	40.1	32	36.5	-	-
Pan-DL (Cheng et al., 2020)	R-50	35.2	-	-	35.5	37.8	32.0
Our baseline	R-50	34.2	35.2	32.8	34.3	34.9	33.4
Our full	R-50	**38.5**	**41.0**	**34.8**	**38.4**	**40.5**	**35.2**

where SSAP uses test-time augmentation and a larger backbone. SMW reports results only on Cityscapes using two independent DeepLabV3+ models and a Mask-RCNN. We outperform it with our approach while still using a simpler model. While Pan-Deeplab outperforms our baseline model, our full training scheme outperforms it on both datasets.

In Figure 3.4 we plot the PQ surrogate (3.5) during fully differentiable training using different numbers of interpolation parameter N in (3.3). Our proposed improvement in the backpropagation scheme of Vlastelica et al. (2019) trains faster and achieves better panoptic quality. In Figure 3.5 we compare our differentiable PQ surrogate (3.5) with the exact PQ metric (3.4) during training. Note that PQ surrogate overestimates exact PQ because we smooth hard thresholding operators. Lastly, we see significant improvement in PQ on evaluation set already after only 24 hours of training with a batch-size of 24 (baseline training took 11 days with 48 batch-size).

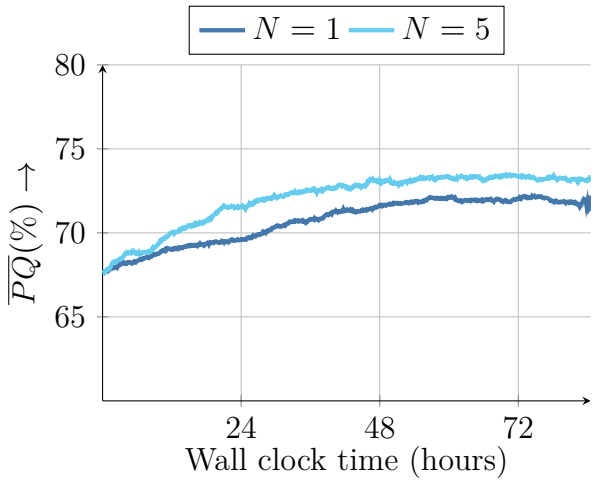


Figure 3.4: Comparison of panoptic quality surrogate loss (higher values better) on Cityscapes for different values of loss interpolation parameter N in (3.3). With $N = 5$ convergence is reached faster, even-though we do not parallelize over N.

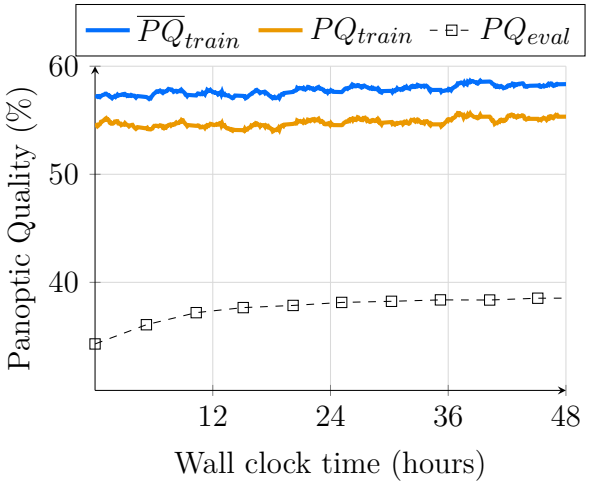


Figure 3.5: Train, evaluation logs on COCO dataset during fully differentiable training. Differentiable surrogate for panoptic quality $\overline{PQ}_{train}$ (3.5) and exact panoptic quality PQ_{train} (3.4) is computed on training set. PQ_{eval} (3.4) is reported on the COCO validation set after every 1000 training iterations.

Ablation study: simpler losses on AMWC. We directly apply loss on semantic class labels x and edge labels y instead of panoptic labels. Since we do not use panoptic labels, this approach does not require transformation to MWC (Alg. 1). The gradients can be computed by perturbing associated semantic costs c_V and edge costs c_E and calling the AMWC solver in the backward pass. Given ground-truth labels x_g, y_g, the losses are

$$L_V = \frac{1}{|V|}\|x - x_g\|_1 \tag{3.7}$$

$$L_E = 1 - \frac{y^T y_g}{y^T y_g + 0.5(y^T(1-y_g) + (1-y)^T y_g)} \tag{3.8}$$

Here the loss on edge labels is based on the F1-score following the approach of SMW (Wolf et al., 2020) to account for class-imbalance. The loss (3.8) is applied separately on each affinity classifier. Afterwards the approach of Vlastelica et al. (2019) can be directly applied to compute gradients except that we use $N = 5$ using the robust backpropagation formula (3.3) for a fair comparison with the panoptic quality surrogate. Lastly, the losses are scaled to put more emphasis on small objects and 'thing' classes in the same way as done for baseline pre-training.

We conduct a comparison on Cityscapes dataset and train using the same setup as for the panoptic quality surrogate loss and use the checkpoint with lowest validation error. Results are given in Table 3.2.

Table 3.2: Comparison of PQ surrogate loss with separate losses on AMWC output

Loss	PQ	PQ_{th}	PQ_{st}
Separate losses	57.8	45.7	66.6
PQ surrogate	**62.1**	**55.1**	**67.2**

We observe that optimizing PQ surrogate gives better performance and using separate losses decreases the performance especially on 'thing' classes. This is due to multiple reasons: (a) The loss applied on affinities cannot perform well w.r.t. PQ because each edge mis-classification is penalized arbitrarily instead of calculating its impact on PQ, (b) a slight localization error in boundary detection is penalized in the same way as errors leading to false merging/splitting of clusters. This issue was also observed by Arganda-Carreras et al. (2015) for 3D instance segmentation.

3.4.4 Limitations

Inference times. Although parallelization can be simply done during training, our approach lacks real-time performance during inference requiring around 2 seconds per image from Cityscapes and 0.3 seconds for COCO datasets resp. In Chapter 4 we will aim to address this shortcoming.

Two stage training. Our training procedure two steps. First we pre-train the network using simpler losses and then finetune with panoptic quality surrogate loss by backpropagating through AMWC. We follow this approach due to computational efficiency, since the combinatorial part takes a significant amount of time. We hope that with better and faster AMWC solvers training can be converted to a single stage in the future. Moreover we avoid finetuning the whole model with panoptic quality surrogate because IoU based metrics are not separable under expectations w.r.t. different images (Berman et al., 2018). To get good estimates of the loss we therefore require larger batch sizes than for simpler losses used in pre-training. This restriction makes it difficult to train all layers due to GPU memory limitations. It would be interesting to train all parameters by backpropagation through the combinatorial solver and forego the need for pre-training possibly on applications with simpler losses and fast combinatorial solvers.

Comparison with transformer based methods. The scope of our work is to evaluate the benefits of backpropagation through AMWC for panoptic segmentation. We employ a traditional CNN based backbone for easier experimentation. Due to this however, modern transformer based approaches (Wang et al., 2020a,b; Jain et al., 2023) outperform our approach. Such approaches also often do not require post-processing for inferring panoptic predictions making them end-to-end trainable. As a drawback however, such methods impose an upper bound on the number of instances. This can be a limitation in case where panoptic predictions contain more than a few hundred objects e.g. in 3D volumes, point clouds etc. As a future work it would be interesting to combine the best of both worlds i.e., employ transformer architectures for learning and differentiable AMWC layer for panoptic segmentation.

3.5 Conclusion

We have proposed a fully differentiable approach for panoptic segmentation incorporating a combinatorial optimization (CO) layer for post-processing and directly minimizing panoptic quality surrogate loss. Our choice has lead to a simple and elegant formulation with a minimal number of hyperparameters. We argue that learning through CO layers is possible and leads to improved performance even with simple and suboptimal solvers. However, backpropagation schemes should be suitably augmented for robustness in this case.

While our work suggests that CO is helpful in neural networks, most solvers (including the ones we used) are sequential and executed on CPU, which limits their applicability. For CO to become a more commonly used layer in neural networks, design of faster solvers is imperative. This will be the focus of the upcoming Chapter 4

4 Massively Parallel Multicut Algorithms

Contents

4.1 Introduction **40**
4.2 Related Work **41**
4.3 Method **42**
4.3.1 Primal: Parallel Edge Contraction 42
4.3.2 Dual: Conflicted Cycles & Message Passing 44
4.3.3 Primal-Dual Updates 46
4.3.4 GPU Implementations 48
4.4 Experiments **49**
4.4.1 Results 51
4.5 Conclusion **53**

In this chapter we propose a highly parallel primal-dual algorithm for the multicut problem Our algorithm consists of three steps executed recursively: (1) Finding conflicted cycles that correspond to violated inequalities of the underlying multicut relaxation, (2) Performing message passing between the edges and cycles to optimize the Lagrange relaxation coming from the found violated cycles producing reduced costs and (3) Contracting edges with high reduced costs through matrix-matrix multiplications. Our algorithm produces primal solutions and lower bounds that estimate the distance to the optimum. We implement our algorithm on GPUs and show resulting one to two orders-of-magnitude improvements in execution speed without sacrificing solution quality compared to traditional sequential algorithms that run on CPUs. We can solve very large-scale benchmark problems with up to $\mathcal{O}(10^8)$ variables in a few seconds with small primal-dual gaps. Our code is available at https://github.com/pawelswoboda/RAMA.

4.1 Introduction

Multicut and its extensions are NP-hard to solve (Bansal et al., 2004; Demaine et al., 2006). Since large problem instances with millions or even billions of variables typically occur, powerful approximate algorithms have been developed (Keuper et al., 2015; Swoboda et al., 2017a; Beier et al., 2014, 2016; Levinkov et al., 2017b). However, even simple heuristics such as GAEC (Keuper et al., 2015) require very large running times for very large instances. In particular, some instances, such as those investigated by Pape et al. (2017) could not be solved in acceptable time (hence ad-hoc decomposition techniques were used). In other scenarios very fast running times are essential e.g., when multicut is used in end-to-end training as done by Song et al. (2019a) and also in Chapter 3. Hence, the need for parallelization arises, preferably on GPUs. The parallelism offered by

GPUs is typically difficult to exploit due to irregular data structures and the inherently sequential nature of most combinatorial optimization algorithms. This makes design of combinatorial optimization algorithms challenging on GPUs. An additional benefit of running our algorithms on GPU is that memory transfers between CPU and GPU are avoided when used in a deep learning pipeline.

Our contribution is a new primal-dual method that can be massively parallelized and run on GPU. This results in faster runtimes than previous multicut solvers while still computing solutions which are similar or better than CPU based solvers in terms of objective. Yet, our approach is rooted in solving a principled polyhedral relaxation and yields both a primal solution and a dual lower bound. In particular, finding primal solutions and approximate dual solving is interleaved such that both components of our algorithm can profit from each other. In more detail, our algorithmic contribution can be categorized as follows

Primal: **Edge Contraction:** Finding a primal solution depends similarly as in GAEC (Keuper et al., 2015) on contracting edges that are highly likely to end up in the same component of the final clustering. To this end, we propose to use a linear algebra approach by expressing edge contractions as sparse matrix-matrix multiplications. This allows us to accelerate edge contraction by exploiting highly parallel matrix-matrix multiplication GPU primitives.

Dual: **Lagrange Relaxation & Message Passing:** To find good edge contraction candidates, we consider approximately solving a relaxation by searching for conflicting cycles, adding them to a Lagrange relaxation and updating the resulting Lagrange multipliers by message passing. We propose a new message passing scheme that is massively parallel thus speeding up the scheme of Swoboda et al. (2017a) by orders of magnitude.

Recursive **Primal-Dual:** We interleave the above operations of finding and solving a Lagrange relaxation and contracting edges, yielding the final graph decomposition. Hence, our algorithm goes beyond classical polyhedral approaches (Swoboda et al., 2017a; Kappes et al., 2011; Nowozin and Jegelka, 2009) that only consider the original graph.

On the experimental side we obtain primal solutions that are of comparable or better quality to those obtained by established high-quality heuristics (Keuper et al., 2015; Lange et al., 2018) in a fraction of the execution time but with additional dual lower bounds that help in estimating the quality of the solutions. We perform experiments on 2D and 3D instance segmentation problems for scene understanding (Cordts et al., 2016) and connectomics (Pape et al., 2017) containing up to $\mathcal{O}(10^8)$ variables.

4.2 Related Work

For a detailed background on related works on multicut algorithms we refer to Sec 2.1.3. We now discuss other related methods for graph clustering. The mutex watershed (Wolf et al., 2020) and its generalizations (Bailoni et al., 2019) are closely related to the greedy

additive edge fixation heuristic for multicut (Levinkov et al., 2017a). The corresponding algorithms can be executed faster than their multicut counterparts on CPU, but are sequential. Fast GPU scheme were proposed for agglomerative clusterings (Auer and Bisseling, 2012). Last, spectral clustering can be implemented on GPU with runtime gains (Jin and JaJa, 2016; Naumov and Moon, 2016). All these approaches however are not based on any energy minimization problem, hence do not come with the theoretical benefits that an optimization formulation offers.

4.3 Method

Recall a *decomposition (or clustering)* of a weighted graph $G = (V, E, c)$ with vertices V, edges E and edge costs $c \in \mathbb{R}^E$ can be obtained by solving the multicut problem (MC)

$$\min_{y \in \mathcal{M}_G} \sum_{ij \in E} c_{ij} y_{ij}. \tag{MC}$$

In summary given an edge uv, positive costs $c_{uv} > 0$ favor the nodes u and v to be in the same component and vice versa. The multicut problem optimizes for an edge labeling which straddles distinct components with minimum cost. The resulting edge label y_{uv} for an edge uv is 1 (resp. 0) if u and v belong to distinct (resp. same) components.

Below we detail the key components of our algorithm: Starting from a graph where each node is a cluster, primal updates consist of edge contractions that iteratively merge clusters by join operations. Dual updates optimize a Lagrange relaxation via message passing to obtain better edge costs and lower bound. Primal and dual updates are interleaved to yield our primal-dual multicut algorithm. We additionally detail how each operation can be done in a highly parallel manner.

4.3.1 Primal: Parallel Edge Contraction

The idea of edge contraction algorithms is to iteratively choose edges with large positive costs. Such edges prefer their endpoints to be in the same component, hence they are contracted and end up in the same cluster. Edge contraction is performed until no contraction candidates are found. The special case of greedy additive edge contraction (GAEC) from Keuper et al. (2015) (Alg. 2.1) chooses in each iteration an edge with maximum edge weight for contraction and stops if each edge in the contracted graph has negative weight. The following Lemma describes the operation of edge contraction.

Lemma 5. *Let an undirected weighted graph $G = (V, E, c)$ and a set of edges $S \subseteq E$ to contract be given. Also let $G' = (V', E', c')$ be the graph obtained after edge contraction.*

(a) The corresponding surjective contraction mapping *$f : V \to V'$ mapping node set V onto the contracted node set V' is up to isomorphism uniquely defined by $f(u) = f(v) \iff \exists uv\text{-path}(V, S)$. The contracted edge set is given by $E' = \{f(u)f(v) : f(u) \neq f(v), uv \in E\}$.*

(b) The edge weights for contracted edges are $c'_{ij} = \sum_{uv \in E : f(u)=i, f(v)=j} c_{uv}$, $\forall ij \in E'$.

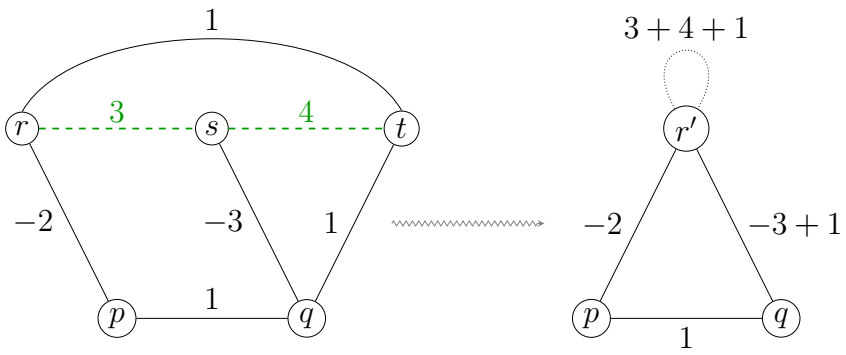


Figure 4.1: Contraction of a graph with contraction set $S = \{rs, st\}$ where vertices r, s and t are merged to form a cluster r'. The corresponding contraction mapping is $f(p) = p, f(q) = q, f(r) = f(s) = f(t) = r'$. Notice that edges qs and qt become parallel edges after contraction and their costs are added. Also notice the presence of self-loop in the contracted graph with cost indicating intra-cluster similarity.

Lemma 5(a) relates the contraction mapping f with the set S of edges to contract. If two nodes have the same value in f then there must be a path between them in graph (V, S). Moreover the edges whose end points are not contracted are preserved in E'. Lemma 5(b) provides the costs of contracted edges that are obtained by summing the costs of parallel edges. An illustration of the lemma is given in Figure 4.1.

In order to perform edge contraction fast we will use a linear algebraic representation that will allow to use highly parallel (sparse) matrix-matrix multiplication. We will perform edge contraction with the help of an edge contraction matrix defined as follows.

Definition 6 (Edge Contraction Matrix)**.** *Given a weighted graph $G = (V, E, c)$ and an edge set $S \subset E$ to contract, let f be the contraction mapping and V' the contracted node set. The edge contraction matrix $K_S \in \{0, 1\}^{V \times V'}$ is defined as* $(K_S)_{uu'} = \begin{cases} 1, & f(u) = u' \\ 0, & \text{otherwise} \end{cases}$.

Lemma 7. *Given a weighted graph $G = (V, E, c)$, an edge set $S \subseteq E$ to contract and an associated edge contraction mapping f*

(a) the adjacency matrix of the contracted graph is equal to $K_S^\top A K_S - \operatorname{diag}(K_S^\top A K_S)$, where $\operatorname{diag}(\cdot)$ is the diagonal part of a matrix,

(b) it holds for the diagonal entry $(K_S^\top A K_S)_{u'u'} = \sum_{uv \in E : u' = f(u) = f(v)} c_{uv}$.

Lemma 7(a) provides a way to compute the contracted graph in parallel by sparse matrix-matrix multiplication. Lemma 7(b) allows to efficiently judge whether the newly formed clusters decrease the multicut objective. Specifically if the diagonal contains all positive terms then the corresponding multicut objective will also decrease after contraction.

A primal update iteration is given in Algorithm 4.1 that performs edge contraction as in Lemma 7(a).

Finding contraction edge set S. A vital step for ensuring a good primal update is selecting the edge set S for contraction in Algorithm 4.1. On one hand, we would like to

Algorithm 4.1: Parallel Edge Contraction (PEC)

Data: Graph $G = (V, E, c)$
Result: Contracted Graph $G' = (V', E', c')$, contraction mapping $f : V \to V'$
1 Compute contraction set $S \subseteq E$
2 Compute adjacency matrix A from G
3 Construct contraction mapping $f : V \to V'$
4 Construct contraction matrix K_S
5 $A' = K_S^\top A K_S - \text{diag}(K_S^\top A K_S)$
6 Compute contracted graph $G' = (V', E', c')$ from A'

choose edges in a conservative manner to avoid erroneous contractions. On the other hand, we need to contract as much edges as possible for efficiency. We propose two approaches allowing us to be at the sweet spot for both criterion as follows.

Maximum matching: Perform a fast maximum matching on the positive edges in using a GPU version of the Luby-Jones handshaking algorithm (Cohen and Castonguay, 2012) and select the matched edges for contraction.

Maximum spanning forest without conflicts: Compute a maximum spanning forest on the positive edges with a fast GPU version of Borůvka's algorithm (Wen-mei, 2011) to find initial contraction candidates. Afterwards, iterate over all negative edges ij, find the unique path between i and j in the forest (if it exists) and remove the smallest positive edge. We make use of GPU connected components (Jaiganesh and Burtscher, 2018) to check for presence of these paths and to compute the final contraction mapping.

Both of the above strategies ensure that the resulting join operation decreases the multicut objective. We first find contraction edges via maximum matching. If not enough edges are found (i.e., fewer than $0.1|V|$), we switch to the spanning forest based approach. Note that if we chose only one largest positive edge for contraction, Algorithm 4.1 specializes to GAEC (Keuper et al., 2015). Since our algorithm depends upon many simultaneous edge contractions for efficiency, we do not use this strategy.

4.3.2 Dual: Conflicted Cycles & Message Passing

Solving a dual of multicut problem (MC) can help in obtaining a lower bound on the objective value and also yields a reparametrization of the edge costs which can help in better primal updates. Our dual algorithm works on the cycle relaxation for the multicut problem (Lemma 3). We present for its solution massively parallel inequality separation routines to search for the most useful violated constraints and efficient dual block coordinate ascent procedure for optimizing the resulting relaxation.

Cycle Inequalities & Lagrange Relaxation. While cycle inequalities (2.2) give us a polyhedral relaxation of the multicut problem (MC), our algorithm will operate on a Lagrangean decomposition that was proposed by Swoboda and Andres (2017). It consists

of two types of subproblems joined together via Lagrange variables: (i) edge subproblems for each edge $e \in E$ and (ii) triangle subproblems (i.e., cycles of length 3) for a subset of triangles $T \subset \binom{E}{3}$. Triangulation of cycles of length more than three is done to get triangles defining the same polyhedral relaxation as the one with all possible cycle inequalities (2.2) without loss of generality (Chopra and Rao, 1993). We define the set of feasible multicuts on triangle graphs as

$$\mathcal{M}_T = \{(0,0,0),(1,1,0),(1,0,1),(0,1,1),(1,1,1)\}, \tag{4.1}$$

which is a special case of (2.2) representing that either all edges are cut/joined or exactly two edges are cut. Given a set of edge and triangle subproblems our Lagrange decomposition is

$$\max_{\lambda} \underbrace{\sum_{uv\in E} \min_{y\in\{0,1\}} c^{\lambda}_{uv} \cdot y + \sum_{t\in T} \min_{y\in\mathcal{M}_T} \langle c^{\lambda}_t, y\rangle}_{=:\mathrm{LB}(\lambda)} \tag{4.2}$$

where the *reparametrized* edge costs $c^{\lambda}_{uv} \in \mathbb{R}$ and triangle costs $c^{\lambda}_t \in \mathbb{R}^3$ for triangle $t = \{ij, jk, ki\} \in T$ are

$$c^{\lambda}_{uv} = c_{uv} + \sum_{t\in T: uv\in t} \lambda_{t,uv} \tag{4.3a}$$

$$c^{\lambda}_t = -(\lambda_{t,ij}, \lambda_{t,jk}, \lambda_{t,ki}) \tag{4.3b}$$

$\mathrm{LB}(\lambda)$ in (4.2) is a lower bound on the cost of the optimum multicut for any λ. The optimum objective value of (4.2) equals that of the polyhedral relaxation of Swoboda et al. (2017a).

Cycle inequality separation. Similar to Lange et al. (2018) we enumerate only the conflicted cycles (see Def. 2) of G for efficiency without loosening the relaxation. A cycle is called a conflicted cycle if it contains exactly one repulsive edge.

Remark 8. *The search for conflicted cycles can be performed in parallel for each $ij \in E^-$ by finding shortest path w.r.t. hop distance between i and j in the graph (V, E^+) making good use of parallelization capabilities of GPUs.*

Dual block coordinate ascent (DBCA). DBCA (a.k.a. message passing) was studied by Swoboda and Andres (2017) for the multicut problem. However, the resulting message passing schemes are not easily parallelizable. The underlying reason for the inherent sequential nature of these schemes is that the effectiveness of the proposed message passing operations depend on the previous ones being executed. We propose a message passing scheme for multicut that is invariant to the message passing schedule, hence allowing parallel computation. Similar to the work of Swoboda and Andres (2017), our scheme iteratively improves the lower bound (4.2) by message passing between edges and triangles.

For each message passing operation we need to compute min-marginals i.e., the difference of optimal costs on subproblems obtained by fixing a specified variable to 1 and 0. For edge costs the min-marginal is just the reparametrized edge cost. For triangle subproblems it is given as follows.

Definition 9 (Marginalization for triangle subproblems)**.** *Let $t \in T$ be a triangle containing an edge e.*

$$m_{t\to e}(c_t^\lambda) = \min_{\substack{y_e=1 \\ y\in\mathcal{M}_T}} \langle c_t^\lambda, y\rangle - \min_{\substack{y_e=0 \\ y\in\mathcal{M}_T}} \langle c_t^\lambda, y\rangle \tag{4.4}$$

is called min-marginal *for triangle t and edge e.*

Algorithm 4.2: Parallel Message Passing (PMP)

Data: Graph $G = (V, E, c)$, triangles T, Lagrange multipliers λ.
Result: Updated Lagrange multipliers λ

```
// Messages from edges to triangles
1  for e ∈ E in parallel do
2      α = c_e^λ
3      for t ∈ T : e ∈ t do
4          λ_{t,e} −= α / |t∈T:e∈t|
5      end
6  end
   // Messages from triangles to edges
7  for t = {ij, jk, ki} ∈ T in parallel do
8      λ_{t,ij} += (1/3) m_{t→ij}(c_t^λ)
9      λ_{t,ik} += (1/2) m_{t→ik}(c_t^λ)
10     λ_{t,jk} += m_{t→jk}(c_t^λ)
11     λ_{t,ij} += (1/2) m_{t→ij}(c_t^λ)
12     λ_{t,ik} += m_{t→ik}(c_t^λ)
13     λ_{t,ij} += m_{t→ij}(c_t^λ)
14 end
```

The message passing algorithm iteratively sets min-marginal to zero first for edge subproblems and then for triangles described in Algorithm 4.2. By sending messages back and forth between subproblems they communicate their local optima and ultimately the min-marginals converge towards agreement (i.e., their corresponding edge labels y are consistent). It was shown by Swoboda et al. (2017a) that each such operation is non-decreasing in the dual objective value, yielding an overall monotonic convergence. Message are passed from edges to triangles in lines 2-5. After this step the reparametrized edge costs c_e^λ become zero. We perform multiple triangle to edge message passing updates (line 8-13) similar to the way it was done by Tourani et al. (2018) that distribute messages uniformly among all triangles which contain that edge. After this operation min-marginals for c_t^λ become zero.

4.3.3 Primal-Dual Updates

While the two building blocks of our multicut solver i.e., edge contraction and cycle separation with message passing can be used in isolation to compute a primal solution

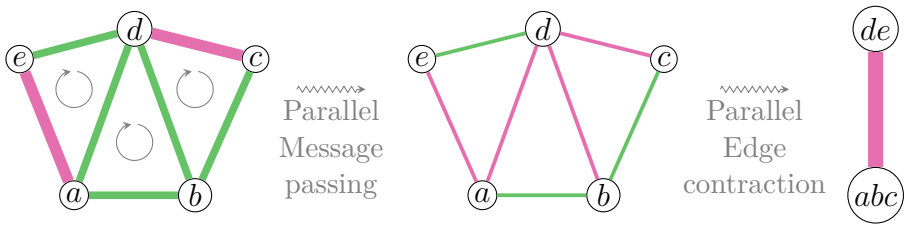


Figure 4.2: Example iteration of our primal-dual multicut solver on a graph with repulsive and attractive edges (best viewed in color). Width of the edges indicate absolute cost. First we detect conflicted cycles (e.g., $\{bc, cd, bd\}$ containing the repulsive edge cd). Next we triangulate conflicted cycles to get triangles (indicated by ↻). Afterwards, dual update reparametrizes edge costs which resolving the conflicted cycles. Lastly, a primal update is done by contracting attractive edges (i.e., $\{ab, bc, de\}$).

and lower bound, we propose an interleaved primal-dual solver in Algorithm 4.3 with an example in Figure 4.2.

Algorithm 4.3: Primal-Dual Multicut

Data: Graph $G = (V, E, c)$
Result: Contraction mapping $f : V \to V'$
// Initialize each node as a separate cluster
1 $f = V \to V,\ f(v) = v \quad \forall v \in V$
2 **while** *G has positive edges without conflicts* **do**
  // Find conflicted cycles (Remark 8)
3 $T = \texttt{Cycle-Separation}(G)$
4 **for** *iter*$= 1, \dots, k$ **do**
    // Do parallel message passing (Alg. 4.2)
5 $\lambda = \texttt{PMP}(G, T)$
    // Reparametrize edge costs
6 $c_e = c_e^\lambda \quad \forall e \in E$ by (4.3a)
7 **end**
  // Do parallel edge contraction (Alg. 4.1)
8 $G, f' = \texttt{PEC}(G)$
  // Update contraction mapping
9 $f(v) = f'(f(v))\ \forall v \in V$
10 **end**

In each iteration we separate cycles and perform message passing to get reparameterized edge costs. We use these reparametrized edge costs to perform parallel edge contraction. This interleaved process continues until no edge contraction candidate can be found. Such scheme has the following benefits

Better edge contraction costs: The reparametrization in line 6 produces edge costs c^λ that are more indicative of whether an edge is contracted or not in the final solution thus yielding better primal updates in line 8. In case the relaxation (4.2) is tight,

the sign of c_e^λ perfectly predicts whether an edge e is separating two clusters or is inside one.

Better cycle separation: For fast execution times we stop cycle separation for cycles greater than a given length (5 in our case). Since cycle separation is performed again after edge contraction, this corresponds to finding longer cycles in the original graph. Such approach alleviates the need to perform a more exhaustive and time-consuming initial search.

Note that a valid lower bound can be obtained from Algorithm 4.3 by recording (4.2) after cycle separation and message passing on the original graph.

4.3.4 GPU Implementations

Edge contraction. We use a specialized implementation for edge contraction using Thrust (Hoberock and Bell, 2010) which is faster than performing it via general sparse matrix-matrix multiplication routines and most importantly has lesser memory footprint allowing to run larger instances. We store the problem graph as an adjacency matrix $A = (I, J, C)$ in COO format, where I, J, C correspond to row indices, column indices and edge costs resp. The pseudocode is given in Algorithm 4.4.

Algorithm 4.4: Parallel Edge Contraction pseudocode

Data: Adjacency matrix $A = (I, J, C)$, Contraction mapping $f : V \to V'$
Result: Contracted adjacency matrix $A' = (I', J', C')$

```
// Assign new node IDs
```
1 $\hat{I}(v) = I(f(v)),\ \forall v \in V$
2 $\hat{J}(v) = J(f(v)),\ \forall v \in V$
3 `COO-Sorting`$(\hat{I}, \hat{J}, C)$
```
// Remove duplicate edges and sum their costs
```
4 $(I', J', C') =$ `reduce_by_key(keys=` $(\hat{I}, \hat{J})$, `values=` $\hat{C}$, `accumulator` $= +)$

Conflicted cycles. For detecting conflicted cycles we use specialized CUDA kernels. The pseudocode for detecting 5-cycles is given in Algorithm 4.5. The algorithm searches for conflicted cycles by parallelizing over repulsive edges. For nodes of each repulsive edge it traverses the neighbors connected by attractive edges $\mathcal{N}^+$. To efficiently check for intersection in Line 2 we store the adjacency matrix in CSR format.

Algorithm 4.5: Parallel Conflicted 5-Cycles pseudocode

Data: Graph $G = (V, E, c)$
Result: Conflicted cycles Y in A

$Y = \varnothing$
// Find attractive paths between repulsive edges
for $v_1v_3 \in \{\mathcal{N}^+(v_0) \times \mathcal{N}^+(v_4) \mid c_{v_0v_4} < 0\}$ *in parallel* **do**
  for $v_2 \in \mathcal{N}^+(v_1) \cap \mathcal{N}^+(v_3)$ **do**
    $Y = Y \cup \{v_0, v_1, v_2, v_3, v_4\}$
  end
end

4.4 Experiments

We evaluate solvers on multicut problems for neuron segmentation for connectomics in the fruit-fly brain (Pape et al., 2017) and unsupervised image segmentation on Cityscapes (Cordts et al., 2016). We use a single NVIDIA Volta V100 (16GB) GPU for our solvers unless otherwise stated and an AMD EPYC 7702 for CPU solvers. Our solvers are implemented using the CUDA (NVIDIA et al., 2021) and Thrust (Hoberock and Bell, 2010) GPU programming frameworks.

Datasets. We have chosen three datasets containing the largest multicut problem instances we are aware of. The instances are made available by Swoboda et al. (2022a).

Connectomics-SP: Contains neuron segmentation problems from the fruit-fly brain (Pape et al., 2017). The raw data is taken from the CREMI-challenge (Funke et al., 2016) acquired by Zheng et al. (2018) and converted to multiple multicut instances by Pape et al. (2017). For this conversion Pape et al. (2017) also reduced the problem size by creating super-pixels. The majority of these instances are different crops of one global problem. There are 3 small ($400000 - 600000$ edges), 3 medium ($4 - 5$ million edges) and 5 large ($28 - 650$ million edges) multicut instances. For the largest problem we use an NVIDIA RTX 8000 (48GB) GPU.

Connectomics-Raw: We use the 3 test volumes (sample A+, B+, C+) from the CREMI-challenge (Funke et al., 2016) segmenting directly on the pixel level without conversion to super-pixels. Conversion to multicut instances is carried out using the framework of Pape (2021). We report results on two types of instances: (i) The three full problems where the underlying volumes have size $1250 \times 1250 \times 125$ with around 700 million edges and (ii) six cropped problems created by halving each volume and creating the corresponding multicut instances each containing almost 340 million edges. For all these instances we use an NVIDIA RTX 8000 (48GB) GPU.

Cityscapes: Unsupervised image segmentation on 59 high resolution images (2048×1024) taken from the Cityscapes validation set (Cordts et al., 2016). Conversion to multicut instances is done by computing the edge affinities produced from Sec 3.4 on a grid graph with 4-connectivity and additional coarsely sampled longer range edges. Each instance contains approximately 2 million nodes and 9 million edges.

Algorithms. As baseline methods we have chosen, to our knowledge, the fastest primal heuristics from the literature.

`GAEC` (Keuper et al., 2015): The greedy additive edge contraction Algorithm 2.1. It is equivalent to Algorithm 4.2 when choosing a single highest edge to contract. We use our own CPU implementation that is around 1.5 times faster than the one provided by the authors.

`KLj` (Keuper et al., 2015): The Kernighan&Lin with joins algorithm performs local move operations which can improve the objective. To avoid large runtimes the output of `GAEC` is used for initialization.

`GEF` (Levinkov et al., 2017a): The greedy edge fixation algorithm is similar to `GAEC` but additionally visits negative valued (repulsive) edges and adds non-link constraints between their endpoints.

`BEC` (Kardoost and Keuper, 2018): Balanced edge contraction, a variant of `GAEC` which chooses edges to contract based on their cost normalized by the size of the two endpoints.

`ICP` (Lange et al., 2018): The iterated cycle packing algorithm searches for cycles and greedily solves a packing problem that approximately solves the multicut dual (4.2).

`P`: Our purely primal Algorithm 4.1 using the maximum matching and spanning forest based edge contraction strategy.

`PD`: Our primal-dual Algorithm 4.3 which additionally makes use of the dual information. We find conflicted cycles up to length 5 on original graph and up to a length of 3 for later iterations on contracted graphs.

`PD+`: Variant of `PD` which always considers conflicted cycles up to a length 5 for reparametrization which can lead to even better primal solutions although with higher runtime.

`D`: Our dual cycle separation algorithm followed by message passing on the original graph via Algorithm 4.2 producing lower bounds.

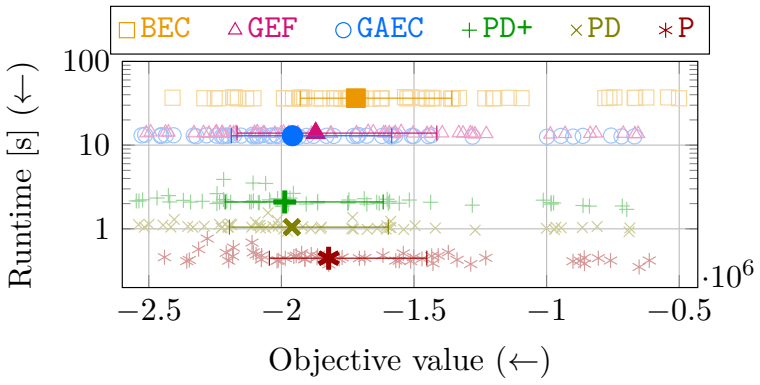


Figure 4.3: Comparison of upper bounds (primal objectives) on *Cityscapes* dataset. Our purely primal algorithm (`P`) is 30× faster than `GAEC` (Keuper et al., 2015) and `GEF` (Levinkov et al., 2017a), although with worse objective values. Incorporating dual information enables our solvers (`PD`, `PD+`) to even surpass the sequential solvers in objective while being faster by an order of magnitude. Error bars mark the 0.25, 0.75-quantile. (`KLj` not shown due to high runtime).

	Connectomics-SP						*Connectomics-Raw*				*Cityscapes*	
	Small (3)		Med. (3)		Large (5)		Crops (6)		Full (3)		(59)	
Method	C(×10^5)	t(s)	C(×10^5)	t(s)	C(×10^5)	t(s)	C(×10^8)	t(s)	C(×10^8)	t(s)	C(×10^6)	t(s)
Primal												
`KLj`	**−1.794**	3.8	**−9.225**	125	†	†	†	†	†	†	−1.858	5e4
`GAEC`	−1.794	0.4	−9.224	4.7	**−1.512**	280	−1.464	570	−2.963	1140	−1.826	13
`GEF`	−1.793	0.7	−9.223	9.0	−1.511	699	−1.458	582	−2.949	1762	−1.743	14
`BEC`	−1.787	0.5	−9.199	5.6	−1.507	309	−1.402	1688	−2.838	4150	−1.613	36
`P`	−1.780	**0.1**	−9.173	**0.6**	−1.505	**6**	−1.430	**9**	−2.895	**19**	−1.711	**0.4**
`PD`	−1.791	0.2	−9.217	1.0	−1.509	13	−1.477	24	−2.981	32	−1.846	1
`PD+`	−1.791	0.3	−9.219	1.4	−1.509	20	**−1.480**	115	**−2.995**	224	**−1.862**	2.2
Dual												
`ICP`	−1.798	0.8	−9.246	11.3	−1.518	1235	−1.507	513	**−3.053**	**1091**	−1.930	41.1
`D`	**−1.797**	**0.2**	**−9.241**	**0.8**	**−1.517**	**13**	**−1.499**	**34**	*	*	**−1.928**	**1.3**

Table 4.1: Comparison of results on all datasets. (C: cost, t(s): time in seconds, †: timed out, *: out of GPU memory). We report average primal and dual costs and runtime over instances within each category. In terms of primal solutions our primal-dual solvers (`PD`, `PD+`) achieve objectives close to or better than sequential solvers while being substantially faster especially on larger instances. Moreover our parallel message passing approach (`D`) gives better lower bounds than `ICP` with up to two orders of magnitude reduction in runtime.

4.4.1 Results

Results on all datasets are given in Table 4.1. On the *Connectomics-SP* dataset we attain primal objectives very close to those produced by `GAEC` (Keuper et al., 2015) but faster by more than an order of magnitude on large instances.

For the *Cityscapes* and *Connectomics-Raw* datasets we achieve even better primal

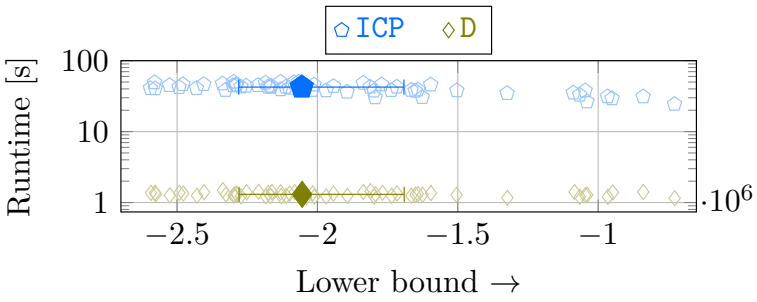


Figure 4.4: Comparison of lower bounds from *Cityscapes* dataset. Our parallel message passing scheme (D) is more than an order of magnitude faster than ICP (Lange et al., 2018) and gives slightly better lower bounds. Error bars mark the 0.25, 0.75-quantile.

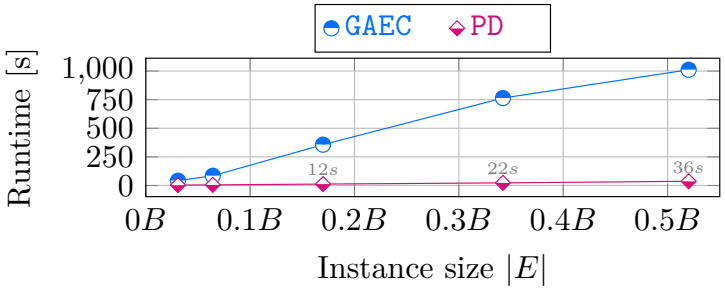


Figure 4.5: Runtime scaling comparison computed on different crops of CREMI test data showing that our PD algorithm scales very well as compared to GAEC (Keuper et al., 2015) w.r.t. increasing problem sizes

solutions than sequential algorithms by incorporating dual information while also being substantially faster. Our best solver (PD+) is more than 10^4 times faster than KLj (Keuper et al., 2015) and produces better solutions. Distributions of runtimes and primal resp. dual objectives for all instances of *Cityscapes* are shown in Figures 4.3 and 4.4. We compare the scaling behaviour of our solver w.r.t. increasing instance sizes in Figure 4.5 showing that our solver scales much more efficiently than GAEC.

Lastly, our dual algorithm (D) produces speedups of up to two orders of magnitude and better lower bounds compared to the serial ICP (Lange et al., 2018), except on the full instances of *Connectomics-Raw* where we run out of GPU memory.

Runtime breakdown. Runtime breakdown of our PD algorithm is given in Table 4.2. Most of the time is spent in finding conflicted cycles which we found to be challenging to implement on GPU while keeping runtime and memory consumption low. Future improvements offer a potential for even better results and speedups by finding longer cycles more efficiently.

Finding S	Contract.	Conf. cycles	Message passing
30%	7%	43%	20%

Table 4.2: Runtime breakdown for PD algorithm on *Cityscapes*

4.5 Conclusion

We have demonstrated that multicut, an important combinatorial optimization problem for machine learning and computer vision, can be effectively parallelized on GPU. Our approach produces better solutions than state of the art efficient heuristics on grid graphs and comparable ones on super-pixel graphs while being faster by one to two orders-of-magnitude. We believe that performance gap on super-pixel graphs is due to a graph structure containing much more (and longer) conflicted cycles. Since our implementation can only find cycles of length up to 5, better implementations that can efficiently handle longer cycles might yield further improvements.

In contrast to CPU algorithms, where execution speed is the limiting factor, for our GPU algorithm, comparatively smaller amount of GPU-memory limits application to even larger instances. We hope that our work will enable more compute intensive applications of multicut, where until now the slower serial CPU codepath has hindered its adoption.

5 Efficient Multicut on Complete Graphs

Contents

5.1 Introduction **54**
5.2 Related Work **55**
5.3 Method **56**
5.3.1 Greedy Additive Edge Contraction 56
5.3.2 Lazy Edge Contraction 60
5.3.3 Varying Affinity Strength 60
5.3.4 Computational Complexity 62
5.4 Experiments **63**
5.4.1 ImageNet Clustering 64
5.4.2 Panoptic Segmentation 65
5.5 Conclusion **68**

In this chapter we propose a graph clustering formulation based on multicut on the complete graph. Our formulation does not need specification of the graph topology as in the original sparse formulation of multicut, making our approach simpler and potentially better performing. In contrast to unweighted correlation clustering, we allow for a more expressive weighted cost structure. In our formulation, the clustering objective is given in a factorized form as the inner products of node feature vectors. This allows for an efficient formulation and inference in contrast to multicut/weighted correlation clustering, which has at least quadratic representation and computation complexity when working on the complete graph. We show how to rewrite classical greedy algorithms for multicut in our dense setting and how to modify them for greater efficiency and solution quality. In particular, our algorithms scale to graphs with tens of thousands of nodes. Empirical evidence on instance segmentation on Cityscapes and clustering of ImageNet datasets shows the merits of our approach.

5.1 Introduction

Graph-based clustering approaches, primarily among them multicut (Chopra and Rao, 1993), are theoretically appealing: They do not need specification of the number of clusters, but infer them as part of the optimization process. They allow for a flexible clustering objective with attractive and repulsive costs between pairs of nodes. They are also theoretically well-understood as optimization problems with intensively studied polyhedral descriptions. Efficient solvers that scale well and give high quality solutions have also been developed.

As a drawback, graph-based clustering approaches need specification of the underlying

graph topology. In practice, this means an additional engineering effort as well as the possibility to not get it right, which would decrease the downstream task performance. Naively circumventing this challenge by using the complete graph is not scalable – the number of edges grows quadratically. One approach to resolve this conundrum is graph structure learning e.g., by extending the work of Kazi et al. (2022), but adds considerable additional complexity.

We propose a method to solve graph clustering efficiently on complete graphs. Our formulation will use the well-known edge-based multicut formulation and only restrict the way edge costs can be computed: they need to be based on inner products of node features. This has two advantages: First, it reduces storage requirements. Instead of storing a full adjacency matrix of edge costs as in multicut, which grows quadratically with the number of nodes, we only need to store a linear number of node features and can compute edge costs on demand. Second, operations needed in multicut algorithms can be made scalable. Instead of operating on the complete graph we can sparsify it adaptively during the solving process. This allows to simulate the workings of multicut algorithms on complete graphs by working on a small subset of it. The key technical ingredient to obtain these sparse subgraphs will be fast nearest neighbor search, for which efficient and scalable implementations exist (Johnson et al., 2019). In effect, this allows us to solve large dense multicut instances in moderate time, which is not possible with existing solvers. In detail, our contribution is as follows:

Formulation: We propose multicut on complete graphs with factorized edge costs as an efficiently representable graph clustering formalism.

Algorithm: We propose scalable algorithms for solving the dense multicut problems, one mimicking exactly the original greedy additive edge contraction (GAEC) algorithm (Keuper et al., 2015), the other a more efficient variant in the spirit of the balanced edge contraction heuristic (Kardoost and Keuper, 2018).

Empirical: We show efficacy in terms of memory and runtime of our solvers and show the merit of using them for image segmentation on Cityscapes and clustering of ImageNet classification dataset.

5.2 Related Work

Algorithms. Multicut algorithms such as the ones from Keuper et al. (2015); Levinkov et al. (2017a) while relatively efficient, scale with the number of edges, making them unsuitable for very large dense graphs. Algorithms for correlation clustering on complete graphs were proposed by Pan et al. (2015); Veldt (2022). However, they only allow unweighted edges. In this chapter we consider efficient algorithms on full graphs and with weighted edges.

K-Means. The K-means problem (Lloyd, 1982) is similar to our approach in that it works directly on feature representations and its objective is based on L_2-distances between features. Similarly to our algorithm, large number of points are handled by efficiently computing kNN-graphs (Qaddoura et al., 2020), thereby reducing run time. In contrast to

multicut, the number of clusters must be given a-priori, while in multicut it is derived as part of the optimization process.

Other clustering approaches. There are a number of other paradigms for clustering. A prominent approach is spectral clustering, in which a weighted graph is given and a clustering is computed with the help of the eigenvectors of the graph Laplacian (Von Luxburg, 2007; Jia et al., 2014). The work of Dhillon et al. (2007) shows connections between weighted k-means and multiple spectral clustering approaches. As for K-means and unlike multicut, spectral clustering requires the number of clusters to be specified.

5.3 Method

Recall a *decomposition (or clustering)* of a weighted graph $G = (V, E, c)$ with vertices V, edges E and edge costs $c \in \mathbb{R}^E$ can be obtained by solving the multicut problem (MC)

$$\min_{y \in \mathcal{M}_G} \sum_{ij \in E} c_{ij} y_{ij}. \tag{MC}$$

The goal of our work is to consider the scenario when the graph G is complete i.e., $E = \{ij : i \in V, j \in V \setminus \{i\}\}$. For large graphs storage and processing of edge costs c becomes prohibitive. To address this issue we instead require as input a feature vector $f_i \in \mathbb{R}^d$ for each node i in V. The edge costs between a pair of nodes i and j can then be measured on-demand through some function $s(f_i, f_j) \to \mathbb{R}$. In this case the multicut problem becomes

$$\min_{y \in \mathcal{M}_G} \sum_{i \in V} \sum_{j \in V \setminus i} s(f_i, f_j) y_{ij}, \tag{DM}$$

which we term as dense multicut problem. An illustration of our formulation is given in Figure 5.1.

In the following we first revisit an algorithm to approximately solve the multicut problem (MC) and show its extensions for the dense multicut problem (DM).

5.3.1 Greedy Additive Edge Contraction

The greedy additive edge contraction (GAEC) scheme (Keuper et al., 2015) as described in Alg. 2.1 computes approximate solution of the multicut problem. It initializes each node as a separate cluster and iteratively contracts a pair of nodes i, j with the largest non-negative cost c_{ij} (if it exists). Let m be the node i and j are contracted to. The edge costs of edges incident to m are

$$c_{ml} = c_{il} + c_{jl}, \quad l \in \mathcal{N}_i \cup \mathcal{N}_j \setminus \{i, j\}, \tag{5.1}$$

where costs of non-existing edges are assumed to be 0 and $\mathcal{N}_i$ corresponds to neighbors of i in graph G. For complete graphs directly applying this algorithm by operating on edge costs is computationally expensive. Moreover since each node is connected to all other nodes ($\mathcal{N}_i = V \setminus \{i\}$), cost updates (5.1) during edge contraction take $\mathcal{O}(|V|)$ instructions.

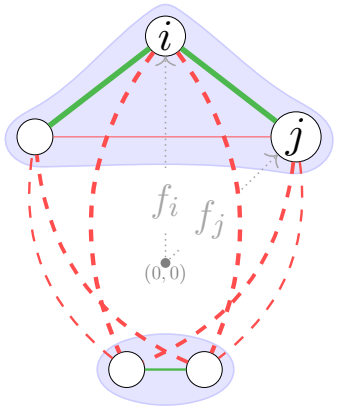


Figure 5.1: Example illustration of dense multicut problem (DM) on 5 nodes. Each node i is associated with a vector $f_i \in \mathbb{R}^2$ and all possible edges between distinct nodes are considered (i.e., the complete graph). The edge cost between a pair of nodes i, j is measured by $\langle f_i, f_j \rangle$ and attractive/repulsive edges are colored green/red. Edge thickness represents absolute edge cost. Also shown is the optimal partitioning to 2 clusters with cut edges denoted by dashed lines.

Contraction on complete graphs. We show how to perform a more efficient (and equivalent) contraction by operating on the node features f by our formulation (DM) for the particular case of $s(\cdot, \cdot)$ defined as

$$s(f_i, f_j) = \langle f_i, f_j \rangle. \tag{5.2}$$

From now on, unless stated otherwise, our edge costs will be given by (5.2).

Lemma 10 (Contraction with node features)**.** *Assume edge costs are measured by* (5.2) *and nodes i and j are contracted to m. Then features of node m given by*

$$f_m = f_i + f_j \tag{5.3}$$

produce contracted edge costs according to (5.1)*.*

Proof. By applying (5.2) for $l \in V$ and comparing with (5.1) we get

$$s(f_m, f_l) = \langle f_m, f_l \rangle = \langle f_i, f_l \rangle + \langle f_j, f_l \rangle = s(f_i, f_l) + s(f_j, f_l)\,. \qquad \square$$

Next we will build on the previous result to devise heuristics for solving dense multicut problem (DM) efficiently.

GAEC for complete graphs. We devise an algorithm which exactly imitates GAEC (Keuper et al., 2015) but is applicable to our formulation on complete graphs (DM). Specifically to make GAEC efficient with node features and a complete graph, we sparsify the original graph G by working on its directed k-nearest neighbors (NN) graph $(V, \mathcal{A})$. The NN graph stores candidate edges for contraction. The arc set $\mathcal{A}$ is populated by nearest neighbor search w.r.t. feature similarity (5.2) and is updated on each edge contraction. We denote outgoing neighbors of i as $\mathcal{N}_i^+ = \{l | (i, l) \in \mathcal{A}\}$ and similarly $\mathcal{N}_i^-$ as incoming neighbors. We write $\mathcal{N}_i^+ \cup \mathcal{N}_j^+$ as $\mathcal{N}_{ij}^+$ and similarly for incoming neighbors. Lastly, the

set $\arg\text{top-k}_{s\in S}\, g(s)$ contains the k elements of S having the largest values of $g(s)$. The complete strategy to obtain a feasible solution of dense multicut problem is described in Algorithm 5.1. It imitates Algorithm 2.1 by iteratively searching and contracting the most attractive edge, but it restricts its search only to the NN graph thereby reducing computation. After contraction, the NN graph is updated (lines 5-8) by only recomputing nearest neighbors of nodes which were affected by the contraction in the NN graph.

Algorithm 5.1: `DenseGAEC`

Data: Node features $f_i, \forall i \in V$; Number of nearest neighbors k
Result: Clusters V

`// Find nearest neighbors of each node`
1 $\mathcal{A} = \{(i,j) \mid i \in V, j \in \arg\text{top-k}_{i' \neq i}\langle f_i, f_{i'}\rangle\}$;
2 **while** $\max_{(u,v)\in\mathcal{A}}\langle f_u, f_v\rangle \geq 0$ **do**
3 $\quad m := (i,j) = \arg\max_{(u,v)\in\mathcal{A}}\langle f_u, f_v\rangle$;
$\quad$ `// Update nodes`
4 $\quad f_m = f_i + f_j$;
5 $\quad V = (V \cup m) \setminus \{i,j\}$;
$\quad$ `// Update nodes having` i,j `as NN`
6 $\quad H = \{(q,r) \mid q \in \mathcal{N}^-_{ij}, r \in \arg\text{top-k}_{l\in V\setminus q}\langle f_q, f_l\rangle\}$;
$\quad$ `// NN of merged node`
7 $\quad H = H \cup \{(m,r) \mid r \in \arg\text{top-k}_{l\in V\setminus m}\langle f_m, f_l\rangle\}$;
$\quad$ `// Add arcs and remove arcs with` i,j
8 $\quad \mathcal{A} = (\mathcal{A} \cup H) \setminus (\{(\cdot,i)\} \cup \{(\cdot,j)\} \cup \{(i,\cdot)\} \cup \{(j,\cdot)\})$;
9 **end**

Proposition 11 (Dense Greedy Contraction)**.** *Algorithm 5.1 always merges a pair of nodes i and j with the largest edge cost i.e.,*

$$(i,j) \in \arg\max_{(u,v)\in\mathcal{A}}\langle f_u, f_v\rangle \implies \langle f_i, f_j\rangle \geq \max_{u,v\neq u}\langle f_u, f_v\rangle. \tag{5.4}$$

Proof. The statement is trivially satisfied before any merge operation is performed since $\mathcal{A}$ is constructed by nearest neighbor search over all nodes in line 1 of the algorithm. We now show that after each merge operation (i.e., after line 8 of the algorithm) the statement (5.4) still holds. We define $Q = m \cup \mathcal{N}^-_{ij}$. Two cases can arise:

Case 1: $\{i,j\} \cap Q \neq \varnothing$**.** Due to nearest neighbor search for all nodes in Q at lines 6 and 7, the statement holds.

Case 2: $\{i,j\} \cap Q = \varnothing$**.** In this case if i is the contracted node m from the last edge contraction operation then $(i,j) \in \mathcal{A}$ due to line 6. If $i \neq m$ then it remains connected to its nearest neighbors either due to the initial NN search at line 1 or the NN update at lines 6 and 7. □

Note that the claim of Prop. 11 does not ensure that the arcset $\mathcal{A}$ will contain all nearest neighbor arcs after contraction. Instead it guarantees that the most attractive edge will always be present in the nearest neighbor graph, foregoing the need to search in

the complete graph. This proves that the Algorithm 5.1 performs locally optimal merges as proposed by Keuper et al. (2015) and is also scalable to large complete graphs. As a downside the algorithm requires costly nearest neighbor search after every edge contraction. Since computing nearest neighbors and contracting edges is not commutative, in the worst case one has to recompute the nearest neighbors on the contracted graph from scratch.

Incremental nearest neighbors. For faster nearest neighbor updates after edge contraction we show how to reuse more of the previously computed nearest neighbors through the following two approaches. First, for all nodes whose nearest neighbors are merging nodes (i.e., line 6 of Alg. 5.1), we check if merged node m is already a nearest neighbor without requiring exhaustive search. Specifically assume a contracting node i was a k-nearest neighbor of some other node $q \in V \setminus i$. Then the merged node m is a k-nearest neighbor of q if $\langle f_q, f_m \rangle \geq \min_{l \in \mathcal{N}_q^+} \langle f_q, f_l \rangle$. This check can be cheaply performed for all such nodes thereby reducing computation. Second, we devise a criterion which can allow to efficiently populate nearest neighbors of the contracted node m.

Proposition 12 (Incremental nearest neighbors)**.** *Let the k-nearest neighbors $\mathcal{N}_i^+, \mathcal{N}_j^+$ of nodes i and j be given. Assume that nodes i, j are merged to form a new node m. Then edge costs between nodes $v \in V \setminus \mathcal{N}_{ij}^+$ and m are bounded from above by*

$$b_{ij} \coloneqq \min_{p \in \mathcal{N}_i^+} \langle f_i, f_p \rangle + \min_{q \in \mathcal{N}_j^+} \langle f_j, f_q \rangle$$

Proof. Since neighbors of i are computed by nearest neighbors search we have for all nodes $p' \notin \mathcal{N}_i^+$

$$\langle f_i, f_{p'} \rangle \leq \min_{p \in \mathcal{N}_i^+} \langle f_i, f_p \rangle,$$

and similarly for node j. Then by definition of v and Lemma 10 we obtain

$$\begin{aligned} \langle f_m, f_v \rangle &= \langle f_i, f_v \rangle + \langle f_j, f_v \rangle \\ &\leq \min_{p \in \mathcal{N}_i^+} \langle f_i, f_p \rangle + \min_{q \in \mathcal{N}_j^+} \langle f_j, f_q \rangle . \end{aligned}$$

□

The above proposition gives an upper bound of feature similarity (i.e., edge cost) of merged node m with all nodes not in $\mathcal{N}_{ij}^+$. Thus if a node in $\mathcal{N}_{ij}^+$ exceeds this upper bound it is more similar to m than all nodes not in $\mathcal{N}_{ij}^+$. This allows to possibly skip recomputing the nearest neighbors of m in Alg. 5.1 (line 7).

Lemma 13. *If*

$$|\{p \in \mathcal{N}_{ij}^+ : \langle f_m, f_p \rangle \geq b_{ij}\}| \geq k \tag{5.5}$$

then k-nearest neighbor of node m given by $\arg\,\text{top-k}_{v \in V \setminus \{i,j,m\}} \langle f_m, f_v \rangle$ *can be chosen as* $\arg\,\text{top-k}_{p \in \mathcal{N}_{ij}^+} \langle f_m, f_p \rangle$.

Proof. Since the elements of $\mathcal{N}_{ij}^+$ already satisfy the bound b_{ij} from Prop. 12 and there are at least k many such elements, the k-nearest neighbors of node m can be taken from $\mathcal{N}_{ij}^+$. □

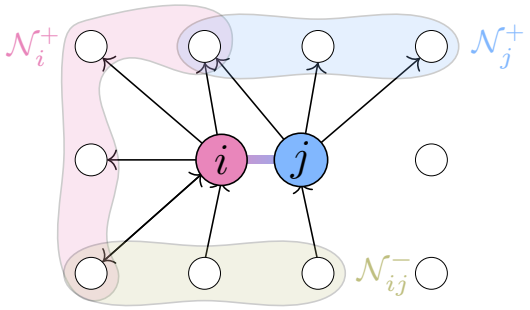


Figure 5.2: Illustration of nearest neighbor graph and an edge ij being contracted. The set $\mathcal{N}_{ij}^+ = \mathcal{N}_i^+ \cup \mathcal{N}_j^+$ is searched first to find nearest neighbors of the merged node efficiently (Prop. 12). The nodes in set $\mathcal{N}_{ij}^-$ need to update their nearest neighbors since their current nearest neighbor nodes i, j are getting contracted. Only the arcs to/from i, j are shown.

Both of these approaches for efficiently updating the NN graph after contraction are used in Alg. 5.2. Additionally in Alg. 5.2 we skip exhaustive search if a node still has p-many nearest neighbors where $p \in [1, k)$. Algorithm 5.2 can be used instead of lines 6 and 7 in Alg. 5.1 for improved performance. See Figure 5.2 for an illustration on nearest neighbor graph and edge contraction update.

5.3.2 Lazy Edge Contraction

We further forego the need for nearest neighbors recomputation after edge contraction by lifting the restriction of performing only greedy moves. This allows to maximally utilize the NN graph: the algorithm performs contractions, including non-greedy ones, until no contraction candidates are present in the NN graph. Specifically we do not perform the exhaustive search in lines 4 and 11 of Alg. 5.2 and only return the nearest neighbors which are easily computable. The NN graph is repopulated as lazily as possible i.e., when no contraction candidates are left. In addition to being more efficient this strategy is reminiscent of the balanced edge contraction approach of Kardoost and Keuper (2018). The authors normalized the edge costs with cluster size of two end-points. These normalized edge costs were used to find the edge to contract. This strategy encouraged consecutive contractions to occur at different regions of the graph. As our lazy approach does not always make the nearest neighbors of the contracted node available thus contractions can only be done to nodes other than the contracted node. This also produces contractions in different regions.

Lastly, we explore efficient methods for approximate nearest neighbor search (Malkov and Yashunin, 2018) for populating the initial NN graph. For later searches we still use exact methods as the search space is reduced due to contractions.

5.3.3 Varying Affinity Strength

Our basic edge costs computed by $\langle f_i, f_j \rangle$ for two features f_i and f_j have one fundamental limitation: Clusters will by default occupy whole quadrants. In other words, whenever two

Algorithm 5.2: Incremental NN update

Data: Contracting nodes i, j; Contracted node m; NN graph $(V, \mathcal{A})$; Node features $f_i, \forall i \in V$; Num. of neighbors k;
Result: Nearest neighbor arcs H to add in $\mathcal{A}$

// NNs of m by Prop. 12
$H = \{(m,l) \mid l \in \mathcal{N}^+_{ij}, \langle f_m, f_l \rangle \geq b_{ij}\}$;
// Keep at most k NN
$H = \arg\text{top-k}_{(m,l)\in H} \langle f_m, f_l \rangle$;
if $H = \varnothing$ **then**
  $H = \{(m,r) \mid r \in \arg\text{top-k}_{l \in V \setminus m} \langle f_m, f_l \rangle\}$;
end
for $q \in \mathcal{N}^-_{ij} \setminus \{i,j\}$ **do**
  // Check if m a NN of q
  if $\langle f_q, f_m \rangle \geq \min_{l \in \mathcal{N}^+_q} \langle f_q, f_l \rangle$ **then**
    $H = H \cup (q,m)$;
  end
  else
    $H = H \cup \{(q,r) \mid r \in \arg\text{top-k}_{l \in V \setminus q} \langle f_q, f_l \rangle)\}$;
  end
end

features have angle lower than 90° they are attractive and will prefer to be in the same cluster, see Figure 5.3. In order to let our formulation favor larger or smaller clusters, we modify our original similarity function $s(\cdot,\cdot)$ by adding an additional term indicated by α-variables:

$$\overline{f}_i = [f_i; \alpha_i], \tag{5.6}$$

$$s(\overline{f}_i, \overline{f}_j) = \langle f_i, f_j \rangle \pm \alpha_i \cdot \alpha_j \,, \tag{5.7}$$

where we choose positive sign for favoring larger clusters and negative for smaller clusters. In our experiments we will set $\alpha_i = \alpha > 0$, with $-$ in (5.7) to prefer many small sized clusters. Moreover we note that our contraction mechanism carries over directly to this extended setting.

Lemma 14. *Aggregating features of the contracted node m by $\overline{f}_m = \overline{f}_i + \overline{f}_j$ is equivalent to setting edge costs as per (5.1) on complete graph.*

Proof. Similar to the proof of Lemma 10 as follows

$$\begin{aligned} s(\overline{f}_m, \overline{f}_l) &= \langle f_m, f_l \rangle \pm \alpha_m \cdot \alpha_l \\ &= \langle f_i + f_j, f_l \rangle \pm (\alpha_i + \alpha_j) \cdot \alpha_l \\ &= \langle f_i, f_l \rangle \pm \alpha_i \cdot \alpha_l + \langle f_j, f_l \rangle \pm \alpha_j \cdot \alpha_l \\ &= s(\overline{f}_i, \overline{f}_l) + s(\overline{f}_j, \overline{f}_l) \,. \end{aligned}$$

□

Large clusters. For preferring larger clusters (corresponding to choosing $+$ in (5.7)), we work directly on the extended feature set $\overline{f}_i = [f_i; \alpha_i]$ and use it in the NN graph.

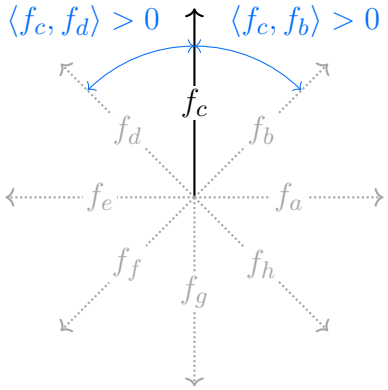


Figure 5.3: Illustration of edge costs between 8 nodes where feature vectors of each node i is in two-dimensional space i.e., $f_i \in \mathbb{R}^2$. If we want each node to be a separate cluster then the edge costs measured by (5.2) are not suitable. This is because there will always be atleast two vectors with positive costs preferring to be in the same cluster. Using a large enough positive value of α with a $-$ in (5.7) this issue can be resolved.

Small clusters. For preferring smaller clusters (corresponding to choosing $-$ in (5.7)), we must modify our algorithms slightly. In order to construct NN graphs we will use two sets of features: First, the query nodes will have their features defined by $\hat{f}_i = [f_i, -\alpha_i]$ and second, the pre-existing nodes $j \in V$ in the graph will keep the same features $\overline{f}_j$ from (5.7). To search for nearest neighbors of node i in the graph V the modified similarity function (5.7) can be implemented by an inner product as

$$s(\overline{f}_i, \overline{f}_j) = \langle \hat{f}_i, \overline{f}_j \rangle \,. \tag{5.8}$$

5.3.4 Computational Complexity

Our basic formulation (DM) with $\alpha = 0$ in (5.7) is shown to be solvable in time $\mathcal{O}(|V|^{d^2})$ by Veldt et al. (2017) through zonotope vertex enumeration (Onn and Schulman, 2001; Stinson et al., 2016). These results are also applicable when choosing $+$ in (5.7). In our experiments having $-$ in (5.7) is vital for obtaining a good clustering, in which case the results of Veldt et al. (2017) are not directly transferable.

Time complexity analysis. Theoretically all of our algorithms have asymptotic time complexity of $\mathcal{O}(d \cdot |V|^3)$. However, empirically we observe our algorithms show quadratic behaviour and get faster through Prop. 12 and lazy contractions. In detail as a worst case scenario of Alg. 5.1, the set $\mathcal{N}^-_{ij}$ in line 6 can be $V \setminus \{i, j\}$. Therefore nearest neighbor search in line 6 has complexity $\mathcal{O}(d \cdot |V|^2)$ making each edge contraction operation quadratic. Overall complexity of the algorithm will then be $\mathcal{O}(d \cdot |V|^3)$. Since, Prop. 12 can fail to produce nearest neighbors thus exhaustive search can be required. Thus asymptotic complexity with Alg. 5.2 for incremental computation remains unchanged.

Note that above complexity analysis assumes that the number of nearest neighbors k is set to 1 which offers very limited potential for incremental nearest neighbor updates. A larger value of k gives much speedup due to Alg. 5.2. A case distinction is provided below

$k = 1$**:** Assume the k-nearest neighbor graph with $k = 1$ before edge contraction has the structure: $V = \{1, 2, ..., n\}$, $\mathcal{A} = \{(2, 1), (3, 1), ...(n, 1)\}$. Thus node 1 is the nearest neighbor of all other nodes. If an edge containing node 1 is contracted it will force all other nodes to recompute their nearest neighbors. Note that there are still $\mathcal{O}(|V|)$ many remaining nodes requiring nearest neighbor update. Due to this worst-case scenario time complexity of one edge contraction becomes quadratic making overall runtime cubic in the number of nodes.

$k \gg 2$**:** Assume the node set after contracting an edge ij is $V' := V \setminus \{i, j\}$. Then each node in V' still has $k - 2$ many nearest neighbors from within V'. In this case nearest neighbor queries only need to be performed between the merged node and nodes in V'. In such case an edge contraction operation can have linear complexity instead of quadratic in the number of nodes. Since we use a value of $k \in [1, 5]$ in all our algorithms utilizing incremental updates, they show such a behaviour. This is also going to be demonstrated empirically in Figure 5.4 in the next section.

5.4 Experiments

We study the benefits of multicut on complete graphs (DM) and compare possible algorithms on the tasks of ImageNet (Deng et al., 2009) clustering and Cityscapes (Cordts et al., 2016) panoptic segmentation. All datasets are made available in Swoboda et al. (2022b).

Algorithms.

`GAEC` (Keuper et al., 2015): The greedy additive edge contraction algorithm from Keuper et al. (2015)(Alg. 2.1) is run on the complete graph where all edge costs are precomputed and then passed to the algorithm.

`RAMA` (Chapter 4): We also compare with the GPU-based multicut solver of Abbas and Swoboda (2022b) as studied in Chapter 4. Similar to `GAEC` we run it on the complete graph. The solver uses dual optimization for better solution quality and also gives lower bounds to the multicut objective. As a drawback it cannot handle large instances due to high memory requirement of complete graphs. We evaluate on an NVIDIA A40 GPU with 48GB of memory.

`DGAEC`: Our Algorithm 5.1 which operates on node features and performs contractions according to Lemma 10. The nearest neighbor graph is updated by exhaustive search after edge contraction. The number of nearest neighbors k is set to 1, a larger value does not benefit since Prop. 12 is not utilized.

`DGAECInc`: Our Algorithm 5.1 which additionally makes use of Alg. 5.2 for incremental neighbor updates after edge contraction. The value of k is set to 5.

`DLAEC`: A variant of our `DGAECInc` where non-greedy moves are also allowed as described in Sec. 5.3.2.

`DAppLAEC`: Another variant of our `DLAEC` where initial nearest neighbors are computed by approximate nearest neighbor search method (Malkov and Yashunin, 2018) through the library of Johnson et al. (2019).

For our dense multicut formulation (DM) on all datasets we set the value of affinity strength α_i in (5.8) to 0.4, preferring small clusters. We do not compare with the randomized algorithm of Veldt et al. (2017) since it does not account for affinity strength with preference on smaller clusters. All CPU algorithms are run on an AMD 7502P CPU with a maximum of 16 threads to allow for faster nearest neighbor search.

5.4.1 ImageNet Clustering

We evaluate clustering of the ImageNet (Deng et al., 2009) validation set containing $50k$ images. Each image in the dataset acts as a node for our dense multicut formulation. The features of each image are computed by a ResNet50 (He et al., 2016a) backbone trained by MoCov3 (Chen et al., 2021) in unsupervised fashion by a constrastive loss on the training split of ImageNet. The features have a dimension of 2048 and are normalized to have unit L_2 norm. We create two problem instances containing $5k$ and $50k$ images by considering 100 and all 1000 classes respectively.

Clustering quality. Before comparing our algorithmic contributions we first test the efficacy of our dense multicut formulation (DM) by comparing its clustering result with k-means (Lloyd, 1982) using the implementation from Pedregosa et al. (2011) and initialization of Arthur and Vassilvitskii (2007). Since k-means requires the number of clusters to be known beforehand we set it to the number of classes in the problem instance. For an additional comparison we also run k-means on the number of clusters given by our dense multicut algorithm. The quality of clustering results are evaluated using normalized mutual information (NMI) and adjusted mutual information (AMI) metrics (Vinh et al., 2010). The results are given in Table 5.1. We observe that although our formulation does not require the number of clusters to be specified, the results are on par with k-means. Additionally the value of affinity strength α does not need to be changed for different problem instances. As compared to k-means our algorithms are much faster especially on the larger instance. The `RAMA` solver from Chapter 4 performs better than all other approaches on the smaller instance but runs out of memory for the larger one. Lastly, our formulation creates more clusters than the number of classes. This is mainly due to presence of outliers in the feature space as the feature extractor is trained without any groundtruth information.

Comparison of algorithms. We compare different algorithms for solving dense multicut problem (DM) for ImageNet clustering in Table 5.2. Firstly, we see that on the smaller instance the GPU based solver `RAMA` gives the best performance. Secondly using incremental nearest neighbor search through Alg. 5.2 gives better run time than exhaustive search. Lastly, our non-greedy algorithms give the best run time among all CPU-based algorithms although with slightly worse objectives.

On the smaller instance, `RAMA` outperforms other algorithms in terms of the objective value (DM) and also gives better clustering quality as compared to k-means. As a drawback `RAMA` cannot handle large dense multicut instances. This shows multicut on

Table 5.1: Quality of clustering on ImageNet validation set. t [s]: compute time in seconds, NMI: normalized mutual information, AMI: adjusted mutual information, # clusters: number of clusters, †: out of GPU memory. For k-means the number of clusters was specified as input.

Method	t [s] ↓	NMI ↑	AMI ↑	# clusters
ImageNet-100 ($\|V\| = 5k$)				
k-means	16	0.42	0.27	100
k-means	32	0.53	0.26	333
`RAMA`	**0.9**	**0.57**	**0.29**	639
`DGAECInc`	42	0.43	0.22	343
`DAppLAEC`	3.2	0.47	0.26	333
ImageNet-1000 ($\|V\| = 50k$)				
k-means	701	0.54	0.2	1000
k-means	1801	**0.61**	0.19	2440
`RAMA`	†	†	†	†
`DGAECInc`	2964	0.49	0.19	2488
`DAppLAEC`	**65**	0.56	**0.26**	2440

complete graphs can be a suitable alternative to k-means. We speculate that algorithmic improvements on top of our proposed algorithms will further improve clustering quality for large graphs.

Lastly, we perform empirical time complexity analysis of our algorithms showing quadratic and subquadratic behaviour in Figure 5.4.

5.4.2 Panoptic Segmentation

We evaluate our method on the task of panoptic segmentation (Kirillov et al., 2019c) on the Cityscapes dataset (Cordts et al., 2016). The panoptic segmentation task consists of assigning a class label to each pixel and partitioning different instances of classes with object categories (e.g., car, person, etc.). We focus on the task of partitioning for which the multicut formulation (MC) has been used by Kirillov et al. (2017) and Abbas and Swoboda (2021) (Chapter 3). The latter work used a carefully crafted graph structure. Our dense multicut (DM) formulation foregoes the need for finding a suitable graph structure. We use the pretrained Axial-ResNet50 (Wang et al., 2021a) network from Yu et al. (2022), made available by Weber et al. (2021) to compute the node features. Specifically, the network computes L_2-normalized, 128-dimensional and 4× downsampled features in its intermediate stages which we use for our study without any training.

For our evaluation we first compute semantic class predictions and then create a dense multicut instance for each semantic category with objects (i.e., car, person, etc.). Such classes are also known as *thing* classes. The goal of the multicut problem is then to

Table 5.2: Comparison of algorithms for solving dense multicut problem on two splits of Imagenet validation set. t [s]: compute time in seconds, Obj: objective value of clustering (DM), †: out of GPU mem. ⋆: no result within 3 hours.

	ImageNet-100		*ImageNet*-1000	
Method	t [s] ↓	Obj ↓	t [s] ↓	Obj ↓
`GAEC`	4.5	-6.84e5	552	**-9.353e7**
`RAMA`	**0.9**	**-6.95e5**	†	†
`DGAEC`	132	-6.84e5	⋆	⋆
`DGAECInc`	42	-6.84e5	2934	**-9.353e7**
`DLAEC`	5	-6.83e5	341	-9.332e7
`DAppLAEC`	3.2	-6.83e5	**65**	-9.332e7

partition all nodes belonging to same semantic class to different objects. This strategy creates a total of 1631 dense multicut problem instances of varying sizes from 500 images of the Cityscapes validation set. The largest problem instance contains around $43k$ nodes. To upsample the clustering back to original image resolution we interpolate the node features back to input image resolution. Afterwards each upsampled node is assigned to the cluster whose mean feature embedding is most similar.

Clustering quality. As a first point of comparison we check whether formulating a multicut problem on the complete graph by (DM) is beneficial as compared to a handcrafted sparse graph structure. We take the sparse graph structure from Abbas and Swoboda (2021) (Chapter 3) as a baseline. Their graph also includes long-range edges for dealing with occlusions leading to about $10 \cdot |V|$ edges in total. We compute the edge costs in this sparse graph in the same way as for our dense formulation and use Alg. 2.1 for computing multicut.

In Table 5.3 we compare the quality of clustering through the panoptic quality metric (Kirillov et al., 2019c). We observe that our dense multicut formulation performs better than multicut on the sparse handcrafted graph. This improvement is significant for classes which can have many instances of the same class within an image (i.e., person, car) thus making the partitioning problem difficult. For classes with large objects (e.g., truck) having more edges does not help since the sparse graph can already capture most inter-pixel relations. On average our dense multicut formulation gives better results than sparse multicut while alleviating the need for designing a graph structure.

Comparison of algorithms . We compare dense multicut algorithms for the panoptic segmentation task in terms of objective value and run time. We were not able to run `RAMA` since the GPU could not store large graphs. The comparison of performance to the remaining algorithms averaged over all problem instances is given in Table 5.4.

In terms of run time, we see that our most naive algorithm `DGAEC` is slower than `GAEC` which directly operates on edge costs. Our other algorithms however, surpass `GAEC` reaching up to an order of magnitude run time improvement with lazy edge contractions and approximate initial nearest neighbors search. In terms of objective value we see slight

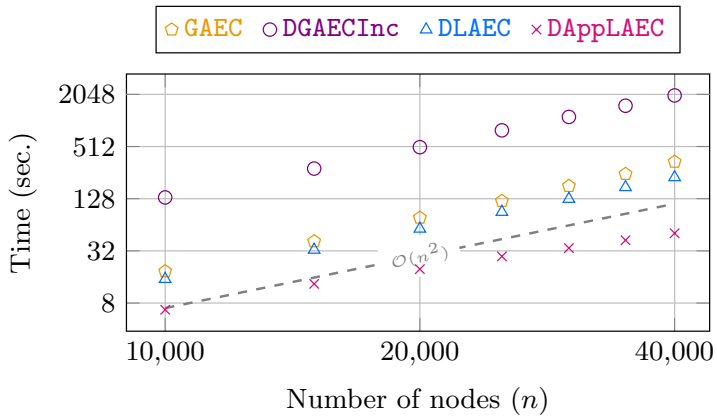


Figure 5.4: Runtime comparison of our algorithms on instances of varying sizes taken from ImageNet val. set by varying number of classes. Both axes are in log-scale. Our algorithms `DGAECInc` and `DLAEC` show quadratic complexity. Our algorithm `DAppLAEC` behaves subquadratically benefiting from approximate nearest neighbor search.

Table 5.3: Comparison of panoptic segmentation on Cityscapes dataset. Multicut on sparse graph of Abbas and Swoboda (2021) is computed by Alg. 2.1. For our dense multicut formulation (DM) we use our `DAppLAEC` algorithm. PQ_{th}: Average panoptic quality of all *thing* classes.

	Panoptic quality (%) ↑	
Category	Sparse multicut	Dense multicut
Person	40.0	**46.9**
Rider	53.0	**54.4**
Car	50.7	**60.5**
Truck	**52.7**	52.3
Bus	**72.1**	71.1
Train	**65.6**	62.9
Motorcycle	**47.0**	46.8
Bicycle	45.7	**46.9**
PQ_{th}	53.3	**55.2**

improvement by our lazy contraction algorithms as compared to the greedy ones.

Sensitivity of affinity strength. In Table 5.5 we study the effect of changing the value of α from (5.7). The results highlight that having $\alpha > 0$ is essential for good clustering quality. Last, we see further improvement if the value of α is set differently for each semantic class. We refer to the Appendix for further results.

Table 5.4: Comparison of algorithms for solving dense multicut problem on Cityscapes validation set. (t [s]): average compute times in seconds, (Obj): average objective value of clustering (DM).

Method	t [s] ↓	Obj ($\times 10^6$) ↓
`GAEC`	7.7	-6.338
`DGAEC`	84.1	-6.338
`DGAECInc`	3.2	-6.338
`DLAEC`	2.1	-6.340
`DAppLAEC`	**1.5**	**-6.341**

Table 5.5: Results of panoptic segmentation via dense multicut with different values of attraction/repulsion strength α in (5.7). PQ_{th}: Avg. panoptic quality over all *thing* classes.

α	0.2	0.3	0.4	0.5	0.6	0.7	0.8
PQ_{th}	54.5	**55.8**	55.2	55.0	54.1	52.0	49.3

5.5 Conclusion

We have demonstrated that optimizing multicut on large complete graphs is possible when using factorized edge costs through inner products of features. We speculate that further algorithmic improvements are possible e.g., by performing dual optimization directly on the node features.

As a potential theoretical advantage our approach sidesteps the need for learning graph structure. This offers a possibility to embed it as a differentiable layer in neural networks similar to Chapter 3.

II

Efficient & Differentiable ILP solver

Background 6

Contents

6.1 Binary Programs **70**
6.2 Lagrangean Decomposition **72**
6.3 Lagrangean Optimization **74**
6.3.1 Dual Block Coordinate Ascent 74
6.3.2 Binary Decision Diagrams 77
6.4 Common Approaches for Structured Prediction **79**
6.4.1 Optimization & Heuristics coupled with Neural Networks . . . 79
6.4.2 Custom Neural Architectures 79

In this part of the thesis we divert our focus to a more general class of methods for solving combinatorial optimization problems. We express the latter as integer linear programs (ILP) and focus on the case where all variables are binary, a typical case in structured prediction tasks from computer vision and machine learning. Although general-purpose ILP solvers have undergone tremendous runtime improvement in past decades, they suffer from scalability issues on large-scale problems. This has necessitated the development of specialized solvers for narrow problem classes (e.g., Markov Random Fields, graph matching, etc.) requiring considerable human effort. The work of Lange and Swoboda (2021) aimed to alleviate this by proposing a general solver for binary ILPs. In the next chapters, we will build on this work to improve its scalability. To this end, we will focus on two key ingredients of massively parallel processing and deep learning. Parallel processing will allow the use of GPUs for achieving faster runtime while learning will help exploit prior solving experience (through training) to improve solver performance on a given problem class. The necessary background and related work are discussed in the upcoming sections, parallel algorithms in Chapter 7, and data-driven solver in Chapter 8.

6.1 Binary Programs

We consider a special case of mixed-integer linear programs where all variables take binary values. The corresponding *0-1 integer linear program* (ILP) is written as

$$\min_x \quad c^\top x \quad \text{s.t.} \quad x \in \bigcap_{j\in[m]} \left\{ x \in \{0,1\}^n \,\middle|\, \sum_{i\in[n]} a_{ij} x_i \leq b_j \right\}, \tag{6.1}$$

where $c \in \mathbb{R}^n$ is the objective vector, $a_{ij}, b_j \in \mathbb{R}$, and m denotes the number of constraints. We provide some examples of 0–1 ILPs used for structured prediction tasks in the following.

Markov Random Fields. The problem of Maximum-A-Posteriori (MAP) inference in Markov Random Fields (MRF) has applications in various domains. Informally the

problem defines a set of nodes where each node indicates its preference of assigning a particular label by some cost. Moreover, a set of factors is given where each factor connects two or more nodes providing preferences about their joint label configurations through higher order costs. The most notable case is where each such factor contains only pairwise relationships. The 0–1 ILP formulation for this case is as follows.

Definition 3 (MAP-MRF as an ILP (Werner, 2007; Savchynskyy, 2019))**.** *Assume a graph $G = (V, E)$ with label space $\mathcal{L}_i$ for each node $i \in V$ is given where each node i takes a label $l_i \in \mathcal{L}_i$ with a unary cost $\theta_i(l_i) \in \mathbb{R}$. Additionally pairwise costs $\theta_{ij}(l_i, l_j) \in \mathbb{R}$ are given for each edge $ij \in E$ taking labels $l_i \in \mathcal{L}_i$ and $l_j \in \mathcal{L}_j$. The 0–1 optimization problem is*

$$\min_{\mu \in \mathcal{U}} \quad \sum_{i \in V} \sum_{l_i \in \mathcal{L}_i} \theta_i(l_i) \cdot \mu_i(l_i) + \sum_{ij \in E} \sum_{l_i \in \mathcal{L}_i} \sum_{l_j \in \mathcal{L}_j} \theta_{ij}(l_i, l_j) \cdot \mu_{ij}(l_i, l_j)\,, \tag{6.2}$$

where

$$\mathcal{U} = \left\{ \mu \;\middle|\; \begin{array}{ll} \sum_{l_i \in \mathcal{L}_i} \mu_i(l_i) = 1 & \forall i \in V \\ \sum_{l_j \in \mathcal{L}_j} \mu_{ij}(l_i, l_j) = \mu_i(l_i) & \forall ij \in E, l_i \in \mathcal{L}_i \\ \sum_{l_i \in \mathcal{L}_i} \mu_{ij}(l_i, l_j) = \mu_j(l_j) & \forall ij \in E, l_j \in \mathcal{L}_j \\ \mu_i(l_i) \in \{0, 1\} & \forall i \in V, l_i \in \mathcal{L}_i \\ \mu_{ij}(l_i, l_j) \in \{0, 1\} & \forall ij \in E, l_i \in \mathcal{L}_i, l_j \in \mathcal{L}_j \end{array} \right\}. \tag{6.3}$$

Graph Matching. Numerous visual computing tasks rely on finding correspondences between two sets of objects. The problem of graph matching, in addition to considering object-object relations also considers relations between pairs of objects. The latter allows to consider the neighborhood of objects in the matching process.

Definition 4 (Graph matching as an ILP (Adams, 1994; Haller et al., 2022))**.** *Assume two sets of nodes V and W where $|W| \geq |V|$ and matching costs $\theta_{vw} \in \mathbb{R}$ for each $v \in V, w \in W$ is given. Additionally, we have costs between pairs of nodes $\theta_{vv',ww'} \in \mathbb{R}$ for distinct $v, v' \in V$ and $w, w' \in W$. Then the 0-1 optimization problem is*

$$\min_{\mu \in \Gamma} \quad \sum_{v \in V} \sum_{w \in W} \theta_{vw} \cdot \mu_{vw} + \sum_{\substack{v,v' \in V \\ v \neq v'}} \sum_{\substack{w,w' \in W \\ w \neq w'}} \theta_{vv',ww'} \cdot \mu_{vv',ww'}\,, \tag{6.4}$$

where

$$\Gamma = \left\{ \mu \;\middle|\; \begin{array}{ll} \sum_{v \in V} \mu_{vw} \leq 1 & \forall w \in W \\ \sum_{w \in W} \mu_{vw} = 1 & \forall v \in V \\ \mu_{vw} = \sum_{v' \in V \setminus v} \sum_{w' \in W \setminus w} \mu_{vv',ww'} & \forall v \in V,\ w \in W \\ \mu_{vw} = \sum_{v' \in V \setminus v} \sum_{w' \in W \setminus w} \mu_{v'v,w'w} & \forall v \in V,\ w \in W \\ \mu_{v,w} \in \{0, 1\} & \forall v \in V, w \in W \\ \mu_{vv',ww'} \in \{0, 1\} & \forall v, v' \in V, w, w' \in W \end{array} \right\}, \tag{6.5}$$

and the first set of constraints is written with an inequality since not every element in W should be matched.

Due to the NP-hard nature of ILPs, relaxations are often employed inside methods for finding solutions (e.g., branch and bound). In addition, relaxations provide lower bounds to the optimal objective allowing us to judge the quality of obtained solutions. In the following, we will discuss one such relaxation method often utilized in structured prediction problems. Before that we will write 0–1 ILP (6.1) for convenience as follows

Definition 5 (Binary Program)**.** *Assume a set of variables $\mathcal{I}$ and an objective vector c in $\mathbb{R}^{\mathcal{I}}$ is given. Moreover we are given a set of constraints $\mathcal{J}$ with corresponding feasible sets $\mathcal{X}_j \subset \{0,1\}^{\mathcal{I}_j}$ for all $j \in \mathcal{J}$ where $\mathcal{I}_j \subseteq \mathcal{I}$ denotes the variables present in constraint j. The corresponding binary program is defined as*

$$\min_x \quad c^\top x \quad \text{s.t.} \quad x_{\mathcal{I}_j} \in \mathcal{X}_j \quad \forall j \in \mathcal{J}\,, \tag{BP}$$

where $x_{\mathcal{I}_j}$ is the restriction of x to variables in $\mathcal{I}_j$.

Remark 15. *The 0-1 ILP (6.1) can be written as (BP) by setting $\mathcal{I} = [n]$, $\mathcal{J} = [m]$, $\mathcal{I}_j = \{i \in [n] \,|\, a_{ij} \neq 0\}$, and $\mathcal{X}_j = \{x \in \{0,1\}^{\mathcal{I}_j} \mid \sum_{i \in \mathcal{I}_j} a_{ij} x_i \leq b_j\}$.*

6.2 Lagrangean Decomposition

Many combinatorial optimization problems including ILPs (6.1) often have a decomposable structure. The feasible set is comprised of a large number of constraints where optimization over a single constraint is typically easy. Lagrangean decomposition (Guignard and Kim, 1987) exploits this structure for solving relaxations of ILPs and has been successfully utilized in a variety of tasks. For example MAP inference in MRF (Kolmogorov, 2006), graph matching (Hutschenreiter et al., 2021), cell tracking (Haller et al., 2020), multiple object tracking (Hornakova et al., 2021). The work of Lange and Swoboda (2021) provides a general approach to tackle binary programs through Lagrangean decomposition. Next, we provide their decomposition strategy and the corresponding Lagrangean dual problem. Before that, we summarize our notation in Table 6.1 with Figure 6.1 containing an example illustration of the decomposition approach.

$\mathcal{I}$, $\mathcal{J}$	Set of variables, constraints respectively
c_i	Cost associated with variable i
x_i	Optimization variable i
$\mathcal{X}_j$	Feasible set of constraint j
$\mathcal{I}_j$	Set of variables in constraint j
$\mathcal{J}_i$	Set of constraints containing variable i
λ_{ij}	Lagrange multiplier for variable i in subproblem j
λ_j	Lagrange multipliers for *all* variables in subproblem j i.e., $(\lambda_{ij})_{i \in \mathcal{I}_j}$
$E_j(\lambda_j)$	Energy of subproblem j i.e., $\min_{x \in \mathcal{X}_j} x^\top \lambda_j$
m_{ij}^β	Energy of subproblem j by setting x_i to β i.e., $\min_{x \in \mathcal{X}_j, x_i = \beta} x^\top \lambda_j$
M_{ij}	Min-marginal difference $m_{ij}^1 - m_{ij}^0$.

Table 6.1: Description of symbols used in Lagrangean decomposition.

Lemma 16 (Lagrangean dual problem)**.** *Let $\mathcal{J}_i = \{j \in \mathcal{J} \mid i \in \mathcal{I}_j\}$ be the set of constraints containing variable i and $\lambda_{ij} \in \mathbb{R}$ be the Lagrange multiplier associated with $i \in \mathcal{I}$, $j \in \mathcal{J}_i$. The energy for subproblem $\mathcal{X}_j$ w.r.t. Lagrangean dual variables $\lambda_j := (\lambda_{ij})_{i \in \mathcal{I}_j} \in \mathbb{R}^{\mathcal{I}_j}$ is written as*

$$E_j(\lambda_j) := \min_{x \in \mathcal{X}_j} x^\top \lambda_j\,. \tag{6.6}$$

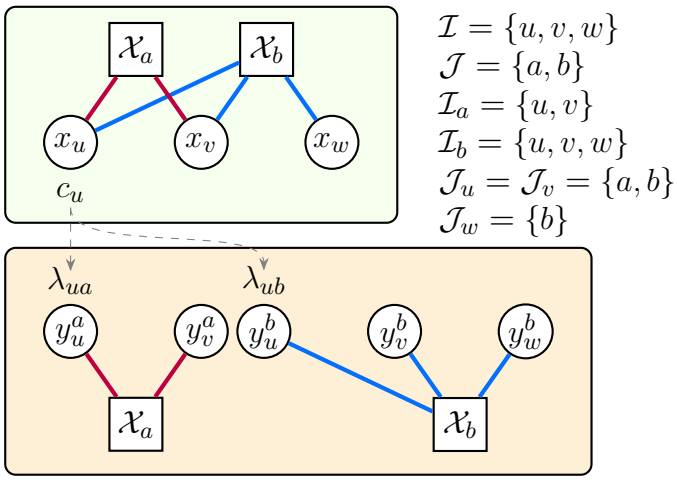


Figure 6.1: Example of a binary program (BP)(*top*) and its Lagrangean decomposition (D)(*bottom*). The binary program contains three variables u, v, w and two constraints a, b. The variables x_u and x_v are duplicated to y_u^a, y_u^b and y_v^a, y_v^b respectively for the decomposition. Costs c of the original problem (BP) constrain the associated Lagrange variables λ in the dual (D). For example λ_{ua} and λ_{ub} should sum to c_u.

Then the Lagrangean dual problem is defined as

$$\max_{\lambda} \quad \sum_{j \in \mathcal{J}} E_j(\lambda_j) \quad \text{s.t.} \quad \sum_{j \in \mathcal{J}_i} \lambda_{ij} = c_i \quad \forall i \in \mathcal{I}, \tag{D}$$

and provides a lower bound to the original problem (BP).

Proof. We introduce an additional set of variables y^j which are independent for each constraint j. The binary program (BP) can be equivalently written as

$$\min_{x,y} \quad c^\top x \quad \text{s.t.} \quad \begin{array}{ll} y^j \in \mathcal{X}_j & \forall j \in \mathcal{J}, \\ y_i^j = x_i & \forall j \in \mathcal{J},\ i \in \mathcal{I}_j, \end{array}, \tag{6.7}$$

where the equality constraints enforce agreement among the variables of each constraint without which the problem can be decoupled. Therefore we introduce a Lagrange multiplier $\lambda_{ij} \in \mathbb{R}$ for each equality constraint and obtain

$$\min_{x, \{y^j \in \mathcal{X}_j\}_j} c^\top x + \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}_j} \lambda_{ij}(y_i^j - x_i) \tag{6.8a}$$

$$= \min_x \sum_i (c_i - \sum_{j \in \mathcal{J}_i} \lambda_{ij}) x_i + \sum_{j \in \mathcal{J}} \min_{y^j \in \mathcal{X}_j} \lambda_j^\top y^j \tag{6.8b}$$

$$= \begin{cases} -\infty & \exists i \in \mathcal{I} : \sum_{j \in \mathcal{J}_i} \lambda_{ij} \neq c_i \\ \sum_{j \in \mathcal{J}} \min_{y^j \in \mathcal{X}_j} \lambda_j^\top y^j & \forall i \in \mathcal{I} : \sum_{j \in \mathcal{J}_i} \lambda_{ij} = c_i \end{cases}. \tag{6.8c}$$

We can restrict to the second non-trivial case of $\sum_{j \in \mathcal{J}_i} \lambda_{ij} = c_i$ for all i thus obtaining the Lagrangean dual. The lower bound property of (D) can be seen by comparing (6.7) with (6.8a). □

If optima of the individual subproblems in (D) agree with each other then the consensus vector obtained from stitching together individual subproblem solutions solves (6.7) and thus also the original problem (BP). In general, (D) is a lower bound on (BP) due to the previous Lemma 16.

6.3 Lagrangean Optimization

Several strategies for optimizing a Lagrangean decomposition are studied in the literature. Since the dual problem (D) is a non-smooth, concave problem (Savchynskyy, 2019) a simple strategy is based on supergradient ascent (Savchynskyy, 2019). Some example works are (Storvik and Dahl, 2000; Komodakis and Tziritas, 2007; Komodakis et al., 2010) for MAP-MRF and (Torresani et al., 2008) for graph matching problems. Although such schemes have provable convergence guarantees they tend to be too slow for practical purposes (Kappes et al., 2013).

Another related class is proximal methods which tackle non-smoothness of the decomposed problems offering faster convergence than subgradient based methods. The MAP-MRF problem has been studied in (Kappes et al., 2012; Ajanthan et al., 2017; Swoboda and Kolmogorov, 2019; Kolmogorov, 2023). Some examples from other areas include a scalable linear programming solver (Applegate et al., 2021) and a specialized linear programming solver for neural network verification (Bunel et al., 2020). Although such methods exhibit good performance in solving relaxations, it can be challenging to decode a solution to the original problem if it contains integrality constraints.

A more popular class of methods is based on block coordinate ascent. Such algorithms optimize the dual problem w.r.t. a subset of variables in each iteration while the rest remain fixed. Although lacking optimality guarantees and the possibility of getting stuck in suboptimal fixed points, these methods outperform their counterparts in terms of efficiency and provide good solutions in practice (Kappes et al., 2013; Haller et al., 2022). Since our work also aims towards efficiency we will aim to optimize the dual problem (D) through such an approach and provide more details in the upcoming section.

6.3.1 Dual Block Coordinate Ascent

An extensive amount of literature exists on dual block coordinate ascent (DBCA) for the MAP-MRF problem. The earliest work in this regard is by Kovalevsky and Koval (1975) studied later by Werner (2007). Some notable approaches include TRW-S (Kolmogorov, 2006), SRMP (Kolmogorov, 2014), MPLP (Globerson and Jaakkola, 2008) and (Werner, 2007; Savchynskyy et al., 2012; Jancsary and Matz, 2011; Meltzer et al., 2012; Wang and Koller, 2013; Johnson et al., 2007; Tourani et al., 2018). Applications in other problems outside of MRF are graph matching (Zhang et al., 2016; Swoboda et al., 2017b, 2019), multicut (Lange et al., 2018; Swoboda and Andres, 2017), multiple object tracking (Hornakova et al., 2021) and cell tracking (Haller et al., 2020). A comparison of algorithms for MAP-MRF (including DBCA) is done in the survey (Kappes et al., 2015) and for graph matching in (Swoboda et al., 2017b; Haller et al., 2022). DBCA algorithms for MAP-MRF are unified by Tourani et al. (2020) where the authors combine strengths of different

methods to create an overall well-performing algorithm across a variety of graph structures. The work of Swoboda et al. (2017a) provides a general framework for understanding and comparing DBCA algorithms for many problem classes. The monograph (Savchynskyy, 2019) provides a detailed background on optimization for MRFs.

Despite the success of such DBCA approaches in solving a variety of problems, they remain specialized warranting rework and human effort each time a new problem class is encountered. A general Lagrange decomposition based scheme was considered by Lange and Swoboda (2021) through binary decision diagrams (BDDs). This approach offered a common framework to design and improve decomposition based ILP algorithms. Due to its generality, we will build upon this work in the subsequent chapters.

A crucial set of quantities of DBCA algorithms for optimizing the Lagrangean (D), are *min-marginals*.

Definition 6 (Min-marginals (Lange and Swoboda, 2021))**.** *For a variable $i \in \mathcal{I}$, constraint $j \in \mathcal{J}_i$ and $\beta \in \{0, 1\}$ the min-marginal is defined as*

$$m_{ij}^{\beta} = \min_{x \in \mathcal{X}_j} x^\top \lambda_j \quad \text{s.t.} \quad x_i = \beta. \tag{MM}$$

Note that we denote min-marginals by m_{ij}^{β} instead of $m_{ij}^{\beta}(\lambda)$ by always assuming the latter unless stated otherwise. For notational convenience let us also define

Definition 7 (Min-marginal differences)**.** *For a variable $i \in \mathcal{I}$ and constraint $j \in \mathcal{J}_i$ the min-marginal difference is computed through* (MM) *as*

$$M_{ij} = m_{ij}^1 - m_{ij}^0, \tag{MD}$$

If $M_{ij} > 0$ then assigning a value of 0 to variable i has a lower cost than assigning a 1 in the subproblem j and vice-versa. Thus, the quantity $|M_{ij}|$ indicates by how much $E_j(\lambda_j)$ increases if x_i is fixed to 1 (if $M_{ij} > 0$), respectively 0 (if $M_{ij} < 0$). Due to this min-marginals provide more information as compared to subgradients and offer faster objective improvement per iteration (Swoboda et al., 2017b). Although in general cases min-marginals are more costly to compute as compared to subgradients however, problem decomposition is often done in a way that they can be computed efficiently. Moreover many constraints occurring in structured prediction tasks naturally allow efficient min-marginal computation. In the following, we provide a short description of the min-marginal averaging algorithm of Lange and Swoboda (2021) for optimizing the dual problem (D).

Definition 8 (Min-marginal averaging)**.** *Assume min-marginal differences M_{ij}* (MD) *for a variable $i \in \mathcal{I}$ with* all *participating subproblems $j \in \mathcal{J}_i$ be given. The min-marginal averaging update for Lagrange variables λ_{ij} for the variable $i \in \mathcal{I}$ and* all *$j \in \mathcal{J}_i$ is defined as*

$$\lambda_{ij} \leftarrow \lambda_{ij} - M_{ij} + \frac{1}{|\mathcal{J}_i|} \sum_{k \in \mathcal{J}_i} M_{ik} \tag{6.9}$$

.

In general terms, the min-marginal averaging update shares preferences of subproblems regarding the assignment of a variable. Subtracting M_{ij} in (6.9) results in subproblem j sending its preferences about variable i, and in the summation these preferences are averaged to get an overall estimate through $\sum_{k\in\mathcal{J}_i} M_{ik}$ normalized by the number of subproblems $\mathcal{J}_i$.

In the following, we show that each min-marginal averaging step guarantees a monotonically non-decreasing sequence of dual objectives (D).

Lemma 17 (Min-marginal averaging guaranteed non-decrease (Lange and Swoboda, 2021))**.** *The min-marginal averaging update (6.9) w.r.t. $i \in \mathcal{I}$ for all $j \in \mathcal{J}_i$ increases the dual objective (D) by the non-negative value*

$$\min\{0, \sum_{k\in\mathcal{J}_i} M_{ik}\} - \sum_{k\in\mathcal{J}_i} \min\{0, M_{ik}\}.$$

Proof. Let $\bar{\lambda}_{ij} = \lambda_{ij} - M_{ij}$ and recall $E_j(\lambda_j) \coloneqq \min_{x\in\mathcal{X}_j} x^\top \lambda_j$ from (D). Then

$$E_j(\bar{\lambda}_j) = \begin{cases} E_j(\lambda_j) - M_{ij} & \text{if } M_{ij} < 0 \\ E_j(\lambda_j) & \text{else} \end{cases} \tag{6.10a}$$

$$= E_j(\lambda_j) - \min\{0, M_{ij}\}. \tag{6.10b}$$

Note that min-marginal difference M_{ij} becomes zero w.r.t. $\bar{\lambda}$. Let $\bar{\bar{\lambda}}_{ij} = \bar{\lambda}_{ij} + \frac{1}{|\mathcal{J}_i|}\sum_{k\in\mathcal{J}_i} M_{ik}$ to obtain

$$E_j(\bar{\bar{\lambda}}_j) = \begin{cases} E_j(\bar{\lambda}_j) + \frac{1}{|\mathcal{J}_i|}\sum_{k\in\mathcal{J}_i} M_{ik} & \text{if } \frac{1}{|\mathcal{J}_i|}\sum_{k\in\mathcal{J}_i} M_{ik} < 0 \\ E_j(\bar{\lambda}_j) & \text{else} \end{cases} \tag{6.11a}$$

$$= E_j(\bar{\lambda}_j) + \frac{1}{|\mathcal{J}_i|} \min\left\{\sum_{k\in\mathcal{J}_i} M_{ik}, 0\right\} \tag{6.11b}$$

Summing up contributions from all subproblems $\mathcal{J}_i$ we get

$$\sum_{j\in\mathcal{J}_i} E_j(\bar{\bar{\lambda}}_j) - E_j(\lambda_j) = \min\{0, \sum_{k\in\mathcal{J}_i} M_{ik}\} - \sum_{k\in\mathcal{J}_i} \min\{0, M_{ik}\} \geq 0. \tag{6.12}$$

□

For optimizing the dual (D) the min-marginal averaging step (6.9) needs to be repeated for all variables. A simple way is choosing some variable arbitrary ordering for carrying out the updates. However, more sophisticated strategies exist which can improve convergence. These include sending averaged min-marginal differences to a subset of variables (Kolmogorov, 2006, 2014), damping min-marginal differences before sending (Werner et al., 2020), etc. Note that although such strategies do not change the underlying LP relaxation and thus also the optimum, their (suboptimal) fixed points and convergence speed can be different. We refer to the studies of (Tourani et al., 2020; Swoboda et al., 2017a) which compare such scheduling choices with their influence on optimization.

6.3.2 Binary Decision Diagrams

To compute min-marginals in a general yet efficient manner *binary decision diagrams* (BDDs) are utilized by Lange and Swoboda (2021). BDDs provide compact encoding for commonly occurring constraints in structured prediction problems. Note that in worst-case scenarios binary programming over a single constraint remains NP-hard (e.g., knapsack constraint). We will utilize BDDs to represent the feasible sets $\mathcal{X}_j$, $j \in \mathcal{J}$ and to compute their min-marginals (MM). BDDs are in essence directed acyclic graphs whose paths between two special nodes (root and terminal) encode all feasible solutions. Specifically, we use reduced ordered Binary Decision Diagrams (Bryant, 1986).

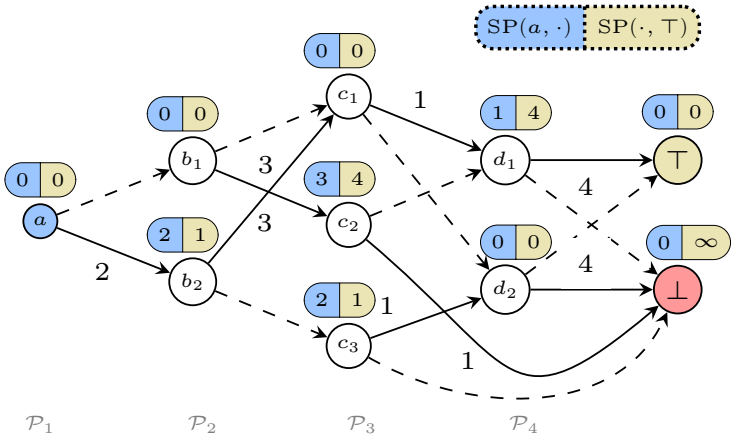


Figure 6.2: Weighted BDD of a subproblem containing variables: $\mathcal{I} = (a, b, c, d)$ with costs λ: $(2, 3, 1, 4)$ resp. and constraint $a - b - c + d = 0$. The shortest path costs from the root node a and the terminal node $\top$ are shown for each node. Here $\mathcal{P}_1 = \{a\}, \mathcal{P}_2 = \{b_1, b_2\}, \mathcal{P}_3 = \{c_1, c_2, c_3\}, \mathcal{P}_4 = \{d_1, d_2\}$, $s^0(c_2) = d_1$ and $s^1(c_2) = \bot$. Dashed arcs have cost 0 as they model assigning a 0 value to the corresponding variable.

Definition 9 (BDD (Abbas and Swoboda, 2022a))**.** *Let an ordered variable set $\mathcal{I} = \{w_1, \dots, w_k\}$ belonging to a constraint be given. A corresponding BDD is a directed acyclic graph $D = (V, A)$ with*

Special nodes: *root node r, terminals $\bot$ and $\top$.*

Outgoing Arcs: *each node $v \in V \backslash \{\top, \bot\}$ has exactly two successors $s^0(v), s^1(v)$ with outgoing arcs $vs^0(v) \in A$ (the zero arc) and $vs^1(v) \in A$ (the one arc).*

Partition: *the node set V is partitioned by $\{\mathcal{P}_1, \dots, \mathcal{P}_k\}$, $\dot\cup_i \mathcal{P}_i = V \backslash \{\top, \bot\}$. Each partition holds all the nodes corresponding to a single variable e.g., $\mathcal{P}_i$ corresponds to variable w_i. It holds that $\mathcal{P}_1 = \{r\}$ i.e., it only contains the root node.*

Partition Ordering: *when $v \in \mathcal{P}_i$ then $s^0(v), s^1(v) \in \mathcal{P}_{i+1} \cup \{\bot\}$ for $i < k$ and $s^0(v), s^1(v) \in \{\bot, \top\}$ for $v \in \mathcal{P}_k$.*

Definition 10 (Constraint Set Correspondence (Abbas and Swoboda, 2022a)). *Each BDD defines a constraint set $\mathcal{X}$ via the relation*

$$x \in \mathcal{X} \Leftrightarrow \begin{array}{c} \exists (v_1, \dots, v_k, v_{k+1}) \in \text{Paths}(V, A) \text{ s.t.} \\ v_1 = r, v_{k+1} = \top, \\ v_{i+1} = s^{x_i}(v_i) \, \forall i \in [k] \end{array} \tag{6.13}$$

Thus each path between root r and terminal $\top$ in the BDD corresponds to some feasible variable assignment $x \in \mathcal{X}$.

Figure 6.2 illustrates BDD encoding of a subproblem representing a linear constraint.

Remark. *In the literature (Bryant, 1986; Knuth, 2011) BDDs have additional requirements, mainly that there are no isomorphic subgraphs. This allows for some additional canonicity properties like uniqueness and minimality. While all the BDDs in our algorithms satisfy the additional canonicity properties, only what is required in Definition 9 is needed for our purposes, so we keep this simpler setting.*

Efficient min-marginal computation. In order to compute min-marginals for subproblems we need to consider weighted BDDs.

Definition 11 (Weighted BDD (Abbas and Swoboda, 2022a)). *A weighted BDD is a BDD with arc costs. Let a function $f(x)$ be defined as*

$$f(x) = \begin{cases} x^\top \lambda_j & x \in \mathcal{X}_j \\ \infty & \text{otherwise} \end{cases}. \tag{6.14}$$

The weighted BDD represents f if it satisfies Def. 10 for the given $\mathcal{X}_j$ and the arc costs for an $i \in [k], v \in \mathcal{P}_i, vw \in A$ are set as $\begin{cases} 0 & w \in s^0(v) \\ \lambda_{ij} & w \in s^1(v) \end{cases}$.

Min-marginals for a variable $i \in \mathcal{I}$ of a subproblem j can be computed by its weighted BDD by calculating shortest path distances from r to all nodes in $\mathcal{P}_i$ and shortest path distances from all nodes in $\mathcal{P}_{i+1}$ to $\top$. We use $\text{SP}(v, w)$ to denote the shortest path distance between nodes v and w of a weighted BDD (see Figure 6.2 for an example). The min-marginals as defined in (MM) can be computed as

$$m_{ij}^\beta = \min_{\substack{v s^\beta(v) \in A \\ v \in \mathcal{P}_i}} \left[\text{SP}(r, v) + \beta \cdot \lambda_{ij} + \text{SP}(s^\beta(v), \top) \right] \tag{6.15}$$

Note that modification of some λ_{ij} will update the costs of all outgoing arcs from BDD nodes in $\mathcal{P}_i$. Importantly, the shortest path costs from root to all nodes preceding $\mathcal{P}_i$ and including $\mathcal{P}_i$ i.e., $\text{SP}(r, v)$ for all $v \in \mathcal{P}_k, k \leq i$, remains unaffected. Same holds true for $\text{SP}(v, \top)$ for all $v \in \mathcal{P}_l, l > i$. The work from Lange and Swoboda (2021) relies on this observation for efficient Lagrangean updates. Specifically shortest path distances are reused and only updated at the affected regions.

6.4 Common Approaches for Structured Prediction

Below we review a general set of approaches (i.e., not necessarily involving combinatorial optimization) for solving structured output prediction tasks in visual computing.

6.4.1 Optimization & Heuristics coupled with Neural Networks

In recent years where deep learning dominates many computer vision tasks, classical algorithms are often employed in conjunction. Such hybrid pipelines can offer the best of both worlds: neural networks utilize training data and predict preferences (e.g., matching costs for key-point matching) for algorithms encoding problem-specific knowledge (e.g., each key-point matches exactly one other). Some recent works in this direction are differentiable pose optimization for multi-view feature matching (Roessle and Nießner, 2023), linear programming solver on top of graph neural networks for tracking (Brasó and Leal-Taixé, 2020; Cetintas et al., 2023), graph matching via integer programming with end-to-end training (Rolínek et al., 2020b), optimal transport for object-centric learning (Zhang et al., 2023), integer programming for shape matching (Windheuser et al., 2011b; Roetzer et al., 2022, 2024), hough voting for panoptic segmentation (Cheng et al., 2020), integer programming for tracking cells in videos paired with 3D convolutional neural networks (Malin-Mayor et al., 2023), etc. For the cell tracking problem, the survey of Maška et al. (2023) compares deep learning versus hand-designed approaches. The authors report superior performance by deep learning based methods for detecting individual cells however, for tracking across frames both deep learning and non-deep learning based methods the performance is similar. Another recent example of deep learning with a graph clustering algorithm is the work by Robert et al. (2024) for 3D panoptic segmentation. The authors demonstrate a reduction in neural network size and faster training as compared to pure neural network based approaches with only a slight degradation in the quality of results.

In other cases, optimization algorithms are employed to obtain supervisory signals for training neural networks (Khoreva et al., 2017; Asano et al., 2019; Wang et al., 2022, 2023). For example normalized cut is utilized by Wang et al. (2023) to infer training targets for image segmentation tasks and optimal transport for training image classification under limited supervision by Asano et al. (2019).

6.4.2 Custom Neural Architectures

Given enough training data neural networks offer a promising way to directly address many structured prediction tasks. Problem-specific details at times can also be incorporated through custom architectures. Examples of such include transformers with learnable queries for panoptic segmentation (Wang et al., 2020a), graph neural networks for tracking (Brasó and Leal-Taixé, 2020) and for feature matching (Lindenberger et al., 2023), 3D convolutional architectures for multi-view stereo (Yao et al., 2018), recurrent layers for optical flow mimicking first-order optimization (Teed and Deng, 2020), transformers for human pose

estimation with learned sub-structures (Geng et al., 2023), etc. A noteworthy example of such methods outside of computer vision is the highly customized architecture for protein folding by Jumper et al. (2021). In some cases however, these custom architectures are highly specialized. For example transformer based panoptic segmentation methods such as (Wang et al., 2020a) cannot explicitly handle a large number of clusters and graph neural networks based tracking approach (Brasó and Leal-Taixé, 2020) needs post-processing via linear programming to satisfy all problem constraints.

7 Massively Parallel 0–1 ILP Algorithms

Contents

7.1 Introduction **81**
7.2 Related Work **82**
7.3 Method **84**
7.3.1 Dual Optimization 84
7.3.2 Primal Rounding 90
7.4 Experiments **91**
7.4.1 Results 93
7.4.2 Limitations 95
7.5 Conclusion **96**

In this chapter we present a massively parallel Lagrange decomposition method for solving 0–1 integer linear programs occurring in structured prediction. We propose a new iterative update scheme for solving the Lagrangean dual and a perturbation technique for decoding primal solutions. For representing subproblems we follow Lange and Swoboda (2021) and use binary decision diagrams (BDDs). Our primal and dual algorithms require little synchronization between subproblems and optimization over BDDs needs only elementary operations without complicated control flow. This allows us to exploit the parallelism offered by GPUs for all components of our method. We present experimental results on combinatorial problems from MAP inference for Markov Random Fields, quadratic assignment, and cell tracking for developmental biology. Our highly parallel GPU implementation improves upon the running times of the algorithms from Lange and Swoboda (2021) by up to an order of magnitude. In particular, we come close to or outperform some state-of-the-art specialized heuristics while being problem-agnostic. Our implementation is available at https://github.com/LPMP/BDD.

7.1 Introduction

Solving integer linear programs (ILP) efficiently on parallel computation devices is an open research question. Done properly it would enable more practical usage of many ILP problems from structured prediction in computer vision and machine learning. Currently, state-of-the-art generally applicable ILP solvers tend not to benefit much from parallelism (Perumalla and Alam, 2021). In particular, linear program (LP) solvers for computing relaxations benefit modestly (interior point) or not at all (simplex) from multi-core architectures. In particular generally applicable solvers are not amenable for execution on GPUs. To our knowledge there exists no practical and general GPU-based optimization routine and only a few solvers for narrow problem classes have been made

GPU-compatible e.g., the works from Shekhovtsov et al. (2016); Tourani et al. (2018); Xu et al. (2020). This, and the superlinear runtime complexity of general ILP solvers has hindered application of ILPs in large structured prediction problems, necessitating either restriction to at most medium problem sizes or difficult and time-consuming development of specialized solvers as observed for the special case of MAP-MRF (Kappes et al., 2015).

We argue that work on speeding up general-purpose ILP solvers has had only limited success so far due to complicated control flow and computation interdependencies. We pursue an overall different approach and do not base our work on the typically used components of ILP solvers. Our approach is designed from the outset to only use operations that offer sufficient parallelism for implementation on GPUs.

We argue that our approach sits on a sweet spot between general applicability and efficiency for problems in structured prediction. Similar to general-purpose ILP solvers (Gurobi Optimization, 2019; CPLEX, 2019), we aim for ease of accessibility. On the other hand we outperform general-purpose ILP solvers in terms of execution speed for large problems from structured prediction and achieve runtimes comparable to hand-crafted specialized CPU solvers. We are only significantly outperformed by specialized GPU solvers. However, development of fast specialized solvers especially on GPU is time-consuming and needs to be repeated for every new problem class (e.g., as done in Chapter 4 for the multicut problem).

Our work builds upon the work of Lange and Swoboda (2021) which utilizes a Lagrange decomposition into subproblems represented by binary decision diagrams (BDD). For optimizing the Lagrangean a sequential as well as a parallel algorithm was proposed. However the parallel algorithm offers only a limited room for parallelization. We improve upon their solver by proposing massively parallelizable GPU amenable routines for both Lagrangean optimization and primal rounding. This results in significant runtime improvements as compared to their approach.

7.2 Related Work

General purpose ILP solvers & parallelism. The most efficient implementation of general-purpose ILP solvers (Gurobi Optimization, 2019; CPLEX, 2019) provided by commercial vendors typically benefit only moderately from parallelism. A recent survey in this direction is done by Perumalla and Alam (2021). The main ways parallelism is utilized in ILP solvers are:

Multiple independent executions. State-of-the-art solvers (Gurobi Optimization, 2019; CPLEX, 2019) offer the option of running multiple algorithms (dual/primal simplex, interior point, different parameters) solving the same problem in parallel until one finds a solution. While easy and worthwhile for problems for which best algorithms and parameters configurations are not known, such a simple approach can deliver parallelization speedups only to a limited degree.

Parallel branch and bound tree traversal. While appealing on first glance, it has been observed by Ralphs et al. (2018) that the order in which a branch and bound tree is traversed is crucial due to exploitation of improved lower and upper bounds and

generated cuts. Consequently, it seems hard to obtain significant parallelization speedups and many recent improvements rely on a sequential execution. A separate line of work (Sofranac et al., 2020) exploited GPU parallelism for domain propagation allowing to decrease the size of the branch and bound tree.

Parallel LP solvers. Interior point methods rely on computing a sequence of solutions of linear systems. This linear algebra can be parallelized for speeding up the optimization (Gondzio and Sarkissian, 2003; Smith et al., 2012). However, for sparse problems sequential simplex solvers still outperform parallelized interior point methods. Also, a crossover step is needed to obtain a suitable basis for the simplex method for reoptimizing for primal rounding and in branch-and-bound searches, limiting the speedup obtainable by this sequential bottleneck. The simplex method is less straightforward to parallelize. The work of Huangfu and Hall (2018) reports a parallel implementation, however current state-of-the-art commercial solvers outperform it with sequentially executed implementations.

Machine learning methods. Recently deep learning based methods have been proposed for choosing variables to branch on (Gasse et al., 2019; Nair et al., 2020) and for directly computing some easy to guess variables of a solution (Nair et al., 2020) or improving a given one (Sonnerat et al., 2021). While parallelism is not the goal of these works, the underlying deep networks are executed on GPUs and hence the overall computation heavy approach is fast and brings speedups. Still, these parallel components do not replace the sequential parts of the solution process but work in conjunction with them, limiting the overall speedup attainable.

The above methods especially suffer in solving very large structured prediction problems in machine learning and computer vision in a few minutes.

Parallel combinatorial solvers. For specialized combinatorial problem classes highly parallel algorithms for GPU have been developed. For Maximum-A-Posteriori inference in Markov Random Fields the works of (Shekhovtsov et al., 2016; Xu et al., 2020) proposed a dual block coordinate ascent algorithm for sparse and Tourani et al. (2018) for dense graphs. For multicut a primal-dual algorithm has been proposed in (Abbas and Swoboda, 2022b) as also covered in Chapter 4. Max-flow GPU implementations have been investigated in (Vineet and Narayanan, 2008; Wu et al., 2012). While some parts of the above specialized algorithms can potentially be generalized, other key components cannot, limiting their applicability to new problem classes and requiring time-consuming design of algorithms whenever attempting to solve a different problem class.

Specialized CPU solvers. There is a large literature of specialized CPU solvers for specific problem classes in structured prediction (see Sec. 6.4). Most of these algorithms however are applicable on narrow problem classes and do not expose enough parallelism to adequately exploit GPUs.

Optimization with binary decision diagrams. Our work builds upon (Lange and Swoboda, 2021). The authors proposed a Lagrange decomposition of ILPs that can be optimized via a sequential dual block coordinate ascent method or a decomposition based approach that can utilize multiple CPU cores. The works of Bergman and Cire (2016, 2018); Lozano et al. (2018) similarly consider decompositions into multiple BDDs

and solve the resulting problem with general-purpose ILP solvers. Optimization of Lagrange decompositions through multi-valued decision diagrams coupled with subgradient methods is investigated in Bergman et al. (2015). An extension for job sequencing was proposed by Hooker (2019) and by Castro et al. (2020) for routing problems. Hybrid solvers incorporating decision diagrams into mixed integer programming solvers were proposed in (Tjandraatmadja and van Hoeve, 2020; González et al., 2020; González et al., 2020). Andersen et al. (2007) utilize decision diagrams for computing a relaxation thus yielding lower bounds. On the other hand a restriction approach for generating approximate solutions was proposed by Bergman et al. (2016a,b).

In contrast to previous BDD-based optimization methods we propose a highly parallelizable and problem agnostic approach that is amenable to GPU computation.

7.3 Method

We briefly introduce the Lagrange decomposition approach to binary ILPs as covered in Sec. 6.1 and propose parallelizable algorithms for both dual and primal optimization. Recall the (primal) binary program (BP)

$$\min_{x\in\{0,1\}^n} \quad c^\top x \quad \text{s.t.} \quad x_{\mathcal{I}_j} \in \mathcal{X}_j \quad \forall j \in \mathcal{J}\,, \tag{BP}$$

its (dual) Lagrangean relaxation (D)

$$\max_{\lambda} \quad \sum_{j\in\mathcal{J}} \min_{x\in\mathcal{X}_j} x^\top \lambda_j \quad \text{s.t.} \quad \sum_{j\in\mathcal{J}_i} \lambda_{ij} = c_i \quad \forall i \in \mathcal{I}, \tag{D}$$

and min-marginal difference M_{ij} (MD) for variable i in subproblem j

$$M_{ij} = \left[\min_{x\in\mathcal{X}_j, x_i=1} x^\top \lambda_j\right] - \left[\min_{x\in\mathcal{X}_j, x_i=0} x^\top \lambda_j\right]. \tag{MD}$$

We will use min-marginal differences for both dual and primal optimization. An example binary program and its subproblem representation via BDDs is illustrated in Figure 7.1.

Next we discuss our parallel update scheme for optimizing the Lagrangean dual (D) followed by our parallel primal rounding algorithm for finding a feasible (not necessarily optimal) of the binary program (BP).

7.3.1 Dual Optimization

We first cover our parallel dual block coordinate scheme for solving the dual problem (D). Next we will make use of second-order information through (parallel) quasi-Newton updates for faster convergence in solving (D).

7.3.1.1 *Parallel Deferred Min-Marginal Averaging*

To exploit GPU parallelism in solving the dual problem we would like to update multiple dual variables in parallel. However, conventional dual update schemes are not friendly for

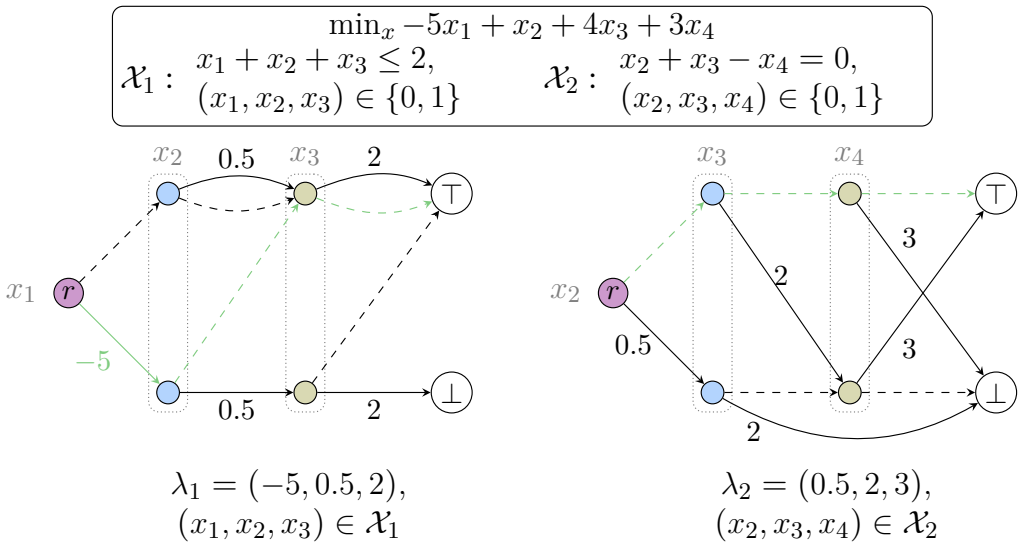


Figure 7.1: Example decomposition of a binary program into two subproblems ($\mathcal{X}_1$, $\mathcal{X}_2$), one for each constraint. Each subproblem is represented by a weighted BDD where solid arcs model the cost λ of assigning a 1 to the variable and dashed arcs have 0 cost which model assigning a 0. All $r - \top$ paths in BDDs encode feasible variable assignments of corresponding subproblems (and $r - \bot$ infeasible). Optimal assignments w.r.t. current (non-optimal) λ are highlighted in green, i.e. $x_1 = 1, x_2 = x_3 = 0$ for $\mathcal{X}_1$ and $x_2 = x_3 = x_4 = 0$ for $\mathcal{X}_2$. Our dual update scheme processes multiple variables in parallel which are indicated in the same color (e.g., x_1, x_2 in $\mathcal{X}_1, \mathcal{X}_2$ resp.).

parallelization. For example the dual update scheme of Lange and Swoboda (2021) for variable i in subproblem j is

$$\lambda_{ij} \leftarrow \lambda_{ij} - M_{ij} + \underbrace{\frac{1}{|\mathcal{J}_i|} \sum_{k \in \mathcal{J}_i} M_{ik}}_{\text{min-marginal averaging}}, \tag{7.1}$$

where M_{ij} is min-marginals difference (MD). This update scheme requires communication between all subproblems $\mathcal{J}_i$ containing variable i for the min-marginal averaging step and thus requires synchronization. To overcome this limitation we propose a novel dual optimization procedure which performs this averaging step on min-marginal differences from the previous iteration $\overline{M}$ as follows

$$\lambda_{ij} \leftarrow \lambda_{ij} - \omega M_{ij} + \frac{1}{|\mathcal{J}_i|} \sum_{k \in \mathcal{J}_i} \overline{M}_{ik}. \tag{7.2}$$

Since $\overline{M}$ was computed in the previous iteration, the above dual updates can be performed in parallel for all subproblems without requiring synchronization. Following the work of Werner et al. (2020) we use a damping factor $\omega \in (0, 1)$ (0.5 in our experiments) to obtain better final solutions.

Our proposed scheme is given in Algorithm 7.1. We iterate over all subproblems j in parallel. For each subproblem, variables are visited in order and min-marginals are

Algorithm 7.1: Parallel Deferred Min-Marginal Averaging (`ParallelMMA`)

Input: Lagrange variables $\lambda_{ij} \in \mathbb{R}\, \forall i \in \mathcal{I}, j \in \mathcal{J}_i$, Constraint sets $\mathcal{X}_j \subset \{0,1\}^{\mathcal{I}_j}\, \forall j \in \mathcal{J}$, Damping factor $\omega \in (0,1]$

1 Initialize deferred min-marginal diff. $\overline{M} = 0$
2 **while** *(stopping criterion not met)* **do**
3 **for** $j \in \mathcal{J}$ *in parallel* **do**
4 **for** $i \in \mathcal{I}_j$ *in ascending order* **do**
5 Compute min-marginal difference M_{ij} (MD) i.e.
$$M_{ij} = \left[\min_{x \in \mathcal{X}_j, x_i = 1} x^\top \lambda_j\right] - \left[\min_{x \in \mathcal{X}_j, x_i = 0} x^\top \lambda_j\right].$$
6 Update dual variables λ_{ij} via (7.2) i.e.,
$\lambda_{ij} \leftarrow \lambda_{ij} - \omega M_{ij} + \frac{1}{|\mathcal{J}_i|} \sum_{k \in \mathcal{J}_i} \overline{M}_{ik}$.
7 **end**
8 **end**
9 Update deferred min-marginal difference $\overline{M} \leftarrow \omega M$
10 Repeat lines 3-6 in descending order of $\mathcal{I}_j$ at line 4
11 **end**
12 **for** $j \in \mathcal{J}$, $i \in \mathcal{I}_j$ **do**
13 Add deferred min-marginal differences: $\lambda_{ij} \leftarrow \lambda_{ij} + \overline{M}_{ij}$
14 **end**

computed and stored for updates in the next iteration (lines 4-5). The current min-marginal difference is subtracted and the one from previous iteration is added (line 6) by distributing it equally among subproblems $\mathcal{J}_i$. At termination (line 13) we perform a min-marginal averaging step to account for the deferred update from last iteration. For stopping criteria we use relative change in dual objective between two subsequent iterations. We initialize the input Lagrange variables by $\lambda_{ij} = c_i / |\,\mathcal{J}_i\,|$, $\forall i \in \mathcal{I}$, $j \in \mathcal{J}_i$.

Proposition 18. *In each dual iteration the Lagrange multipliers along with the deferred min-marginals can be used to satisfy dual feasibility and the dual lower bound* (D) *is non-decreasing.*

Proof.
Feasibility of iterates We prove

$$\sum_{j \in \mathcal{J}_i} \lambda_{ij} + \omega M_{ij} = c_i \tag{7.3}$$

just before line 9 in Algorithm 7.1. We do an inductive proof over the number of iterates w.r.t. iterations t.

$k = 0$**:** Follows from $\overline{M} = 0$ and the uniform distribution of costs at initialization.

$k > 0$**:** Let $\lambda(k-1)$, $M(k-1)$, $\overline{M}(k-1)$ be previous iterations' Lagrange multipliers, min-marginals differences and (deferred) min-marginal differences. Also let $\lambda(k)$,

$M(k)$ and $\overline{M}(k)$ be the same from current iteration just before line 9. Note that $\overline{M}(k) = M(k-1)$. It holds that

$$\sum_{j \in \mathcal{J}_i} [\lambda_{ij}(k) + \omega M_{ij}(k)] \tag{7.4a}$$

$$= \sum_{j \in \mathcal{J}_i} \left[\lambda_{ij}(k-1) - \omega M_{ij}(k) + \sum_{l \in \mathcal{J}_i} \left(\frac{\omega}{|\mathcal{J}_i|} \overline{M}_{il}(k) \right) + \omega M_{ij}(k) \right] \tag{7.4b}$$

$$= \sum_{j \in \mathcal{J}_i} [\lambda_{ij}(k-1) + \omega M_{ij}(k-1)] \tag{7.4c}$$

$$= c_i \,. \tag{7.4d}$$

Non-decreasing lower bound. In order to prove that iterates have non-decreasing lower bound we will consider an equivalent lifted representation in which proving the non-decreasing lower bound will be easier.

Lifted representation. Introduce λ_{ij}^{β} for $\beta \in \{0, 1\}$ and the subproblems

$$E(\lambda_j^1, \lambda_j^0) = \min_{x \in \mathcal{X}_j} x^\top \lambda_j^1 + (1-x)^\top \lambda_j^0 \tag{7.5}$$

Then (D) is equivalent to

$$\max_{\lambda^1, \lambda^0} \sum_{j \in \mathcal{J}} E(\lambda_j^1, \lambda_j^0) \text{ s.t. } \sum_{j \in \mathcal{J}_i} \lambda_{ij}^{\beta} = \beta \cdot c_i \tag{7.6}$$

We have the transformation from original to lifted λ

$$\lambda \mapsto (\lambda^1 \leftarrow \lambda, \lambda^0 \leftarrow 0) \tag{7.7}$$

and from lifted to original λ (except a constant term)

$$(\lambda^1, \lambda^0) \mapsto \lambda^1 - \lambda^0 \,. \tag{7.8}$$

It can be shown that the lower bounds are invariant under the above mappings and feasible λ for (D) are mapped to feasible ones for (7.6) and vice versa.

The update rule line 6 in Algorithm 7.1 for the lifted representation can be written as

$$\lambda_{ij}^{\beta} \leftarrow \lambda_{ij}^{\beta} - \omega \cdot \min(m_{ij}^{\beta} - m_{ij}^{1-\beta}, 0) + \omega \cdot \min(\overline{m}_{ij}^{\beta} - \overline{m}_{ij}^{1-\beta}, 0) \tag{7.9}$$

It can be shown that (7.9) and line 6 in Algorithm 7.1 are corresponding to each other under the transformation from lifted to original λ.

Continuation of non-decreasing lower bound. Define

$$\gamma_{ij}^{\beta} = \lambda_{ij} - \omega \cdot \min(m_{ij}^{\beta} - m_{ij}^{1-\beta}, 0) \,. \tag{7.10}$$

Then $E(\gamma_j^1, \gamma_j^0) = E(\lambda_j^1, \lambda_j^0)$ due to the definition of min-marginals. Define next

$$\delta_{ij}^{\beta} = \gamma_{ij}^{\beta} + \omega \cdot \min(\overline{m}_{ij}^{\beta} - \overline{m}_{ij}^{1-\beta}, 0) \,. \tag{7.11}$$

Then $E(\delta_j^1, \delta_j^0) \geq E(\gamma_j^1, \gamma_j^1)$ since $\delta \geq \gamma$. This proves the claim. Another side-benefit of the lifted update scheme (7.9) is that evaluating $E(\lambda_j^1, \lambda_j^0)$ during the course of optimization in Algorithm 7.1 always gives a lower bound to the true dual objective calculated after accounting for deferred min-marginal differences.

□

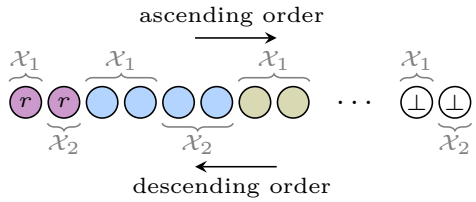


Figure 7.2: Arrangement of BDD nodes in GPU memory for the ILP in Figure 7.1. For ascending order in Alg. 7.1 we proceed from root to terminal nodes and vice versa for descending.

Similar to other dual block coordinate ascent schemes Algorithm 7.1 can get stuck in suboptimal points (Werner et al., 2020; Werner, 2007). However, as seen in our experiments these are usually not far away from the optimum.

Efficient min-marginal computation. For efficient min-marginal computation in Algorithm 7.1 we reuse shortest path distances (6.15) as also discussed in Section 6.1. Specifically, for computing min-marginals in lines 4-5 of Alg. 7.1 we use Alg. 7.2 for ascending variable order and Alg. 7.3 for descending variable order in line 10.

Algorithm 7.2: Forward Pass Min-Marginal Computation

1 **for** $v \in \mathcal{P}_i$ **do**
2 $\quad \mathrm{SP}(r,v) = \min\left\{ \min_{u:s^0(u)=v} \mathrm{SP}(r,u), \min_{u:s^1(u)=v} \mathrm{SP}(r,u) + \lambda_i \right\}$
3 **end**
4 Compute m_i^β via (6.15)

Algorithm 7.3: Backward Pass Min-Marginal Computation

1 **for** $v \in \mathcal{P}_{i+1}$ **do**
2 $\quad \mathrm{SP}(v,\top) = \min\left\{ \mathrm{SP}(s^0(v),\top), \mathrm{SP}(s^1(v),\top) + \lambda_{i+1} \right\}$
3 **end**
4 Compute m_i^β via (6.15)

Efficient GPU implementation. In addition to solving all subproblems in parallel, we also exploit parallelism within each subproblem during shortest path updates. Specifically in Alg. 7.2, we parallelize over all $v \in \mathcal{P}_i$ and perform the min operation atomically. Similarly in Alg. 7.3 we parallelize over all $v \in \mathcal{P}_{i+1}$ but without requiring atomic update.

To enable fast GPU memory access via memory coalescing we arrange BDD nodes in the following fashion. First, all nodes within a BDD which belong to the same partition $\mathcal{P}$ (thus corresponding to same variable) are laid out consecutively. Secondly, across different BDDs, nodes are ordered w.r.t. increasing hop distance from their corresponding root nodes. Such arrangement for the ILP in Figure 7.1 is shown in Figure 7.2.

7.3.1.2 *Quasi-Newton updates via L-BFGS*

We interleave quasi-Newton update steps via limited memory BFGS (L-BFGS) algorithm (Nocedal, 1980; Liu and Nocedal, 1989) with the parallel min-marginal averaging steps (Alg. 7.1). This allows to harness second-order information in the optimization process and empirically aids in faster convergence especially for large instances. An iteration of our algorithm for optimizing (D) is given in Algorithm 7.4.

Algorithm 7.4: Quasi-Newton update with Min-Marginal Averaging

Input: Lagrange variables λ, Estimate of Hessian inverse $\hat{H}$, Previous step size γ,

$g_j(\lambda) \coloneqq \arg\min_{x \in \mathcal{X}_j} x^\top \lambda_j, \quad \forall j \in \mathcal{J}$ // Supergradient of dual obj. (D)

$\hat{d} = \hat{H} g(\lambda)$ // Compute update direction

$d_{ij} = \hat{d}_{ij} - \frac{1}{|\mathcal{J}_i|} \sum_{k \in \mathcal{J}_i} \hat{d}_{ik}, \forall i \in \mathcal{I}, j \in \mathcal{J}_i$ // Make update direction feasible (D)

$\gamma \leftarrow$ `FindStepSize`(λ, d, γ)

$\lambda \leftarrow \lambda + \gamma \cdot d$ // Apply update

$\lambda \leftarrow$ `ParallelMMA`(λ) // One iteration of Alg. 7.1

$\hat{H} \leftarrow$ `L-BFGS`$(\hat{H}, g(\lambda))$ // Update Hessian estimate

return $\lambda, \hat{H}, \gamma$

Procedure `FindStepSize`(λ, d, γ)

  $\gamma_m = \gamma$

  $E_{init} = E(\lambda + \gamma d)$ // Compute initial objective (D)

  for $t = 1, \ldots, K$ **do**

    if $E(\lambda + \gamma d) \leq E_{init}$ **then**

      $\gamma = \underline{\alpha} \gamma$ // Decrease step size

    else

      $\gamma = \overline{\alpha} \gamma$ // Increase step size

    if $E(\lambda + \gamma d) \geq E(\lambda + \gamma_m d)$ **then**

      $\gamma_m = \gamma$

    if $E(\lambda + \gamma d) - E_{init} \geq \Delta_{min}$ **then**

      break

  end

  return γ_m

In detail, we first compute a subgradient of the objective by finding a minimizing assignment for each subproblem. Next, we calculate the search direction by multiplying the subgradient with an estimate of the inverse Hessian. However, naively updating the Lagrange variables in this search direction does not preserve dual feasibility (D). To address this, we modify the update direction d to ensure that $\sum_j d_{ij}$ is zero for all i. Subsequently, a suitable step size is determined that provides sufficient improvement in the objective. Using this step size, we perform the quasi-Newton update, followed by a forward and backward parallel min-marginal averaging iteration of Alg. 7.1. Finally, the estimate of the inverse Hessian is updated for use in subsequent iterations. In essence, our algorithm combines two distinct update schemes for optimizing the Lagrangean dual (D). It employs a first-order update through parallel min-marginal averaging (PMMA), which

guarantees non-decreasing objective values. The L-BFGS update leverages second-order information to facilitate faster convergence towards the optimum. Empirically, we observe this hybrid scheme to be significantly faster than either L-BFGS or min-marginal averaging alone.

Step size selection. The strategy for step size selection for quasi-Newton updates is given in Alg. 7.4 (line 9). In detail, the step size is increased by a factor $\overline{\alpha} > 1$ if non-negative improvement is found in the objective but it is below the threshold Δ_{min} and decrease the step size by $0 < \underline{\alpha} < 1$ if we find non-improvement in the objective. The step size search is done for at most K many iterations for efficiency reasons. If the final step size does not yield an improvement in the objective we do not perform the quasi-Newton update in the current iteration. The updated value of step size is also used as an initial estimate in the next iteration. For ensuring sufficient ascent we check improvement in the objective in a relative sense by setting $\Delta_{min} = 10^{-6} \cdot [E(\lambda(1)) - E(\lambda(0))]$ where $\lambda(0)$ denotes Lagrange variables at the start of dual optimization and $\lambda(1)$ after first invocation of Alg. 7.4. Rest of the parameters are set as $K = 5$, $\underline{\alpha} = 0.8$, $\overline{\alpha} = 1.1$. For L-BFGS updates we store past 5 iterates and use the scheme of Liu and Nocedal (1989) with our lightweight line search strategy.

7.3.2 Primal Rounding

In order to obtain a primal solution to (BP) from an approximative solution to (D) we propose a GPU friendly primal rounding scheme based on cost perturbation. We iteratively change costs in a way that variable assignments across subproblems agree with each other. If all variables agree by favoring a single assignment, we can reconstruct a primal solution (not necessarily the optimal). Instead of only using variable assignments of all subproblems we use min-marginal differences (MD) as they additionally indicate how strongly a variable favors a particular assignment.

Algorithm 7.5 details our method. We iterate over all variables in parallel and check min-marginal differences. If for a variable i all min-marginal differences indicate that the optimal solution is 0 (resp. 1) Lagrange variables λ are increased (resp. decreased) leaving even more certain min-marginals differences for these variables. This step imitates variable fixation as done in branch-and-bound, however we only perform soft fixation implicitly through cost perturbation. In case min-marginal differences are equal we randomly perturb corresponding dual costs. Lastly, if min-marginals differences indicate conflicting solutions we compute total min-marginal difference and decide accordingly. In the last two cases we add more perturbation to force towards non-conflicts. For faster convergence we increase the perturbation magnitude after each iteration.

Note that the modified λ variables via Alg. 7.5 need not be feasible for the dual problem (D). Although our primal rounding algorithm is not guaranteed to terminate, in our experiments a solution was always found in less than 150 iterations.

Remark. *A similar strategy to the primal heuristic presented here was given by Wedelin (1995b,a) for a limited class of problems (binary constraint matrix). This was later generalized by Bastert et al. (2010) where different types of constraints are handled as special cases. Our BDD representation however, can handle a variety of constraints directly*

Algorithm 7.5: Perturbation Primal Rounding

Input: Lagrange variables $\lambda_{ij} \in \mathbb{R}\, \forall i \in \mathcal{I}, j \in \mathcal{J}_i$, Constraint sets $\mathcal{X}_j \subset \{0,1\}^{\mathcal{I}_j}\, \forall j \in \mathcal{J}$, Initial perturbation strength $\delta \in \mathbb{R}_+$, perturbation growth rate α

Output: Feasible labeling $x \in \{0,1\}^n$

Compute min-marginal differences M_{ij} for all $i \in \mathcal{I}$, $j \in \mathcal{J}_i$ (MD)
while $\exists i \in \mathcal{I}$ *and* $j \neq k \in \mathcal{J}_i$ s.t. $\text{sign}(M_{ij}) \neq \text{sign}(M_{ik})$ **do**
  for *variables* $i \in \mathcal{I}$ *in parallel* **do**
    Sample r uniformly from $[-\delta, \delta]$
    if $M_{ij} > 0\ \forall j \in \mathcal{J}_i$ **then**
      `// Variables in all subproblems prefer 0 assignment, increase cost`
      $\lambda_{ij} \mathrel{+}= \delta \quad \forall j \in \mathcal{J}_i$
    end
    else if $M_{ij} < 0\ \forall j \in \mathcal{J}_i$ **then**
      `// Variables in all subproblems prefer 1 assignment, decrease cost`
      $\lambda_{ij} \mathrel{-}= \delta \quad \forall j \in \mathcal{J}_i$
    end
    else if $M_{ij} = 0\ \forall j \in \mathcal{J}_i$ **then**
      `// Variables have no preference, perturb randomly`
      $\lambda_{ij} \mathrel{+}= r \cdot \delta \quad \forall j \in \mathcal{J}_i$
    end
    else
      `// Variables have different preferences, decide by majority decision`
      Compute total min-marginal difference: $M_i = \sum_{j \in \mathcal{J}_i} M_{ij}$
      $\lambda_{ij} \mathrel{+}= \text{sign}(M_i) \cdot |r| \cdot \delta \quad \forall j \in \mathcal{J}_i$
    end
  end
  Increase perturbation: $\delta \leftarrow \delta \cdot \alpha$
  Reoptimize dual problem (D) on perturbed λ via Alg. 7.1 or Alg. 7.4
  Recompute min-marginals $M_{ij}\, \forall i, j$ w.r.t. optimized λ
end

and allows for parallel processing on GPU.

7.4 Experiments

We show effectiveness of our solver against a state-of-the-art ILP solver (Gurobi Optimization, 2019), the BDD-based solver (Lange and Swoboda, 2021) which runs on CPU and specialized CPU solvers for specific problem classes. We have chosen a variety structured prediction binary ILPs from the literature the we know of and are publicly available. Our results are computed on a single NVIDIA RTX 8000 (48GB) GPU. For CPU solvers we use AMD EPYC 7702 CPU with 16 threads.

Datasets. Our benchmark problems obtained from (Swoboda et al., 2022b) can be categorized as follows.

Cell tracking: Instances from Haller et al. (2020) which we partition into small and large instances as also done by Lange and Swoboda (2021).

Graph matching: Quadratic assignment problems (often called graph matching in the literature) for correspondence in computer vision by Torresani et al. (2008) (`hotel`, `house`) and for developmental biology by Kainmueller et al. (2014) (`worms`).

Markov Random Field (MRF): Several datasets from the OpenGM benchmark (Kappes et al., 2015), containing both small and large instances with varying topologies and number of labels. We have chosen the datasets *color-seg-n4*, *color-seg-n8* and *object-seg*.

QAPLib: The widely used benchmark dataset of Burkard et al. (1997) for quadratic assignment problems used in the combinatorial optimization community. We partition QAPLib instances into small (up to 50 vertices) and large (up to 128 vertices) instances. Conversion to ILP is done via (2.7)-(2.16) of Loiola et al. (2007)

Shape matching (SM): Problems for 3D shape matching from Roetzer et al. (2024) released by Swoboda et al. (2022b). The underlying shapes are from the works of Magnet et al. (2022) and Li et al. (2021). We evaluate on instances capturing a more realistic scenario of cross category shape matching (DT4D-Inter). The underlying shapes are non-isometrically deformed and contain from 500 to 800 triangles. The results of `Gurobi` are not reported as it takes more than an hour to solve an instance (in most cases).

Algorithms. We compare results on the following algorithms.

`Gurobi`: The commercial ILP solver (Gurobi Optimization, 2019) as reported in Lange and Swoboda (2021). The dual simplex algorithm is used for all datasets except *QAPLib*. On *QAPLib* dataset the barrier method is faster than dual simplex on small instances. However for large instances it needs more than an hour to report any result. Therefore we compare both algorithms and report results of the better performing one for each instance.

`BDD-CPU`: BDD-based min-marginal averaging approach of Lange and Swoboda (2021). The algorithm runs on CPU with 16 threads for parallelization. Primal solutions are rounded using their BDD-based depth-first search scheme.

`Specialized solvers`: State-of-the-art problem specific solver for each dataset. For *cell tracking* the solver from Haller et al. (2020). For *graph matching* the fastest solvers from recent benchmark by Haller et al. (2022) i.e., `dd-ls3` for *house*, *hotel* and `fm-bca` for *worms*. For *MRF* we take `TRWS` solver by Kolmogorov (2006).

`FastDOG`: Our approach where for the GPU implementation we use the CUDA (NVIDIA et al., 2021) and Thrust (Hoberock and Bell, 2010) programming frameworks. For rounding primal solutions with Algorithm 7.5 we set $\delta = 1.0$ and $\alpha = 1.2$ and perform dual optimization for a maximum of 500 iterations (Alg. 7.1, line 20). For constructing BDDs out of linear (in)equalities we use the same approach as for `BDD-CPU`.

`FastDOG-QN`: Our approach where we additionally use quasi-Newton updates via Algorithm 7.4 for dual optimization. For primal recovery we use same strategy as `FastDOG` except we run dual optimization for at most 100 iterations (instead of 500) for dual optimization on perturbed costs.

For *MRF*, parallel algorithm from Tourani et al. (2018) exist however `TRWS` is faster on the sparse problems we consider. While we are aware of even faster purely primal heuristics (Komodakis and Tziritas, 2007; Boykov et al., 2001) for *MRF* they do not optimize a convex relaxation and hence do not provide lower bounds. Hence we have chosen solvers which optimize an equivalent LP relaxation or similar Lagrange decomposition (D) and are thus directly comparable.

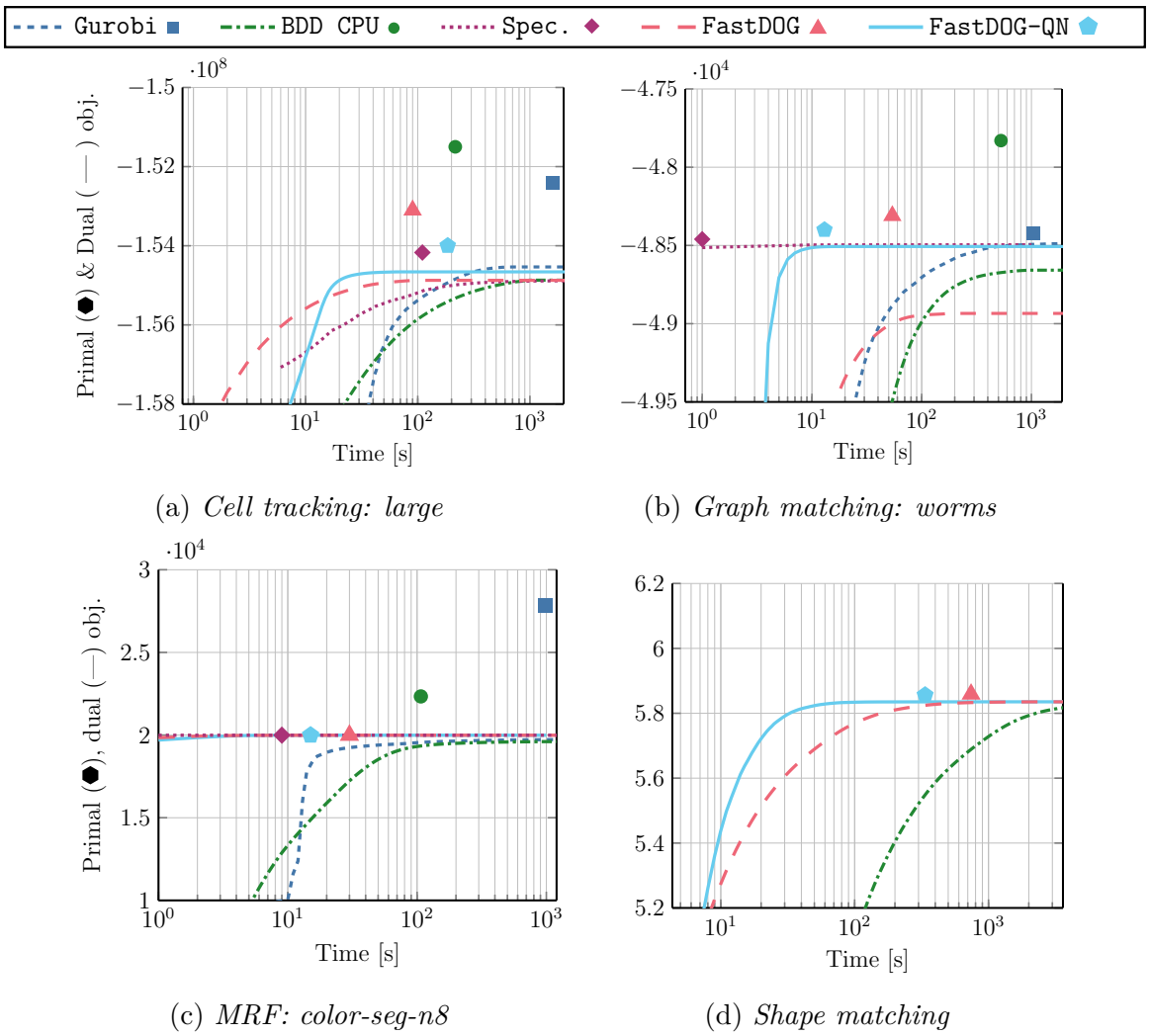
7.4.1 Results

	Cell tracking		*Graph matching*			*MRF*			*QAPLib*		*SM*
	Small	*Large*	*Hotel*	*House*	*Worms*	*C-seg-n4*	*C-seg-n8*	*O-seg*	*Small*	*Large*	
# instances	10	5	105	105	30	9	9	5	105	29	15
$n_{max}(\times 10^6)$	1.2	10	0.3	0.3	1.5	1.2	1.4	0.69	3	49	14
$m_{max}(\times 10^6)$	0.2	2.3	0.05	0.05	0.2	4.2	8.3	2.2	0.24	2	11
obj. multiplier	10^6	10^8	10^3	10^3	10^4	10^4	10^4	10^4	10^6	10^5	1
Dual objective (lower bound) ↑											
`Gurobi`	**−4.382**	**−1.545**	**−4.293**	**−3.778**	−4.849	1.9757	1.9729	3.1311	**8.585**	0.2	-
`BDD-CPU`	−4.387	−1.549	**−4.293**	**−3.778**	−4.878	1.9643	1.9631	3.1248	3.675	81	5.82
`Specialized`	−4.385	−1.551	**−4.293**	**−3.778**	**−4.846**	**2.0012**	**1.9991**	**3.1317**	-	-	-
`FastDOG`	−4.387	−1.549	**−4.293**	**−3.778**	−4.893	2.0011	1.9990	**3.1317**	3.747	89	5.83
`FastDOG-QN`	−4.385	−1.547	**−4.293**	**−3.778**	−4.850	2.0011	**1.9991**	**3.1317**	4.701	**174**	**5.83**
Primal objective (upper bound) ↓											
`Gurobi`	**−4.382**	−1.524	**−4.293**	**−3.778**	−4.842	2.8464	2.7829	14.981	51.86	1431	-
`BDD-CPU`	−4.337	−1.515	**−4.293**	**−3.778**	−4.783	2.1781	2.2338	3.1525	52.39	1452	-
`Specialized`	−4.361	−1.531	**−4.293**	**−3.778**	**−4.851**	**2.0012**	**1.9991**	**3.1317**	-	-	-
`FastDOG`	−4.376	−1.541	**−4.293**	**−3.778**	−4.831	2.0016	1.9995	3.1322	43.30	**1376**	5.86
`FastDOG-QN`	−4.377	**−1.544**	**−4.293**	**−3.778**	−4.839	2.0016	1.9995	3.1341	**35.68**	1427	**5.86**
Runtimes [s] ↓											
`Gurobi`	**1**	1584	4	7	1048	980	1337	1506	3948	6742	-
`BDD-CPU`	14	216	6	12	528	107	218	232	357	5952	2788
`Specialized`	1.5	**90**	1	1	**1**	**9**	30	**3**	-	-	-
`FastDOG`	13	110	**0.2**	**0.4**	54	**9**	13	39	137	6928	741
`FastDOG-QN`	14.8	185	0.6	1.4	13	**9**	19	64	**101**	**2561**	**336**

Table 7.1: Results comparison on all datasets where the values are averaged within a dataset. Numbers in bold highlight the best performance. n_{max}, m_{max}: Maximum number of variables, constraints in the category and obj. multiplier represents postscaling of dual and primal objectives for recovering their original values.

In Table 7.1 we show aggregated results over all instances of each specific benchmark dataset. Runtimes are taken w.r.t. computation of both primal and dual bounds.

In Figure 7.3 we show averaged convergence plots for various solvers. In general we offer a very good anytime performance producing at most times and in general during



(a) *Cell tracking: large* (b) *Graph matching: worms*

(c) *MRF: color-seg-n8* (d) *Shape matching*

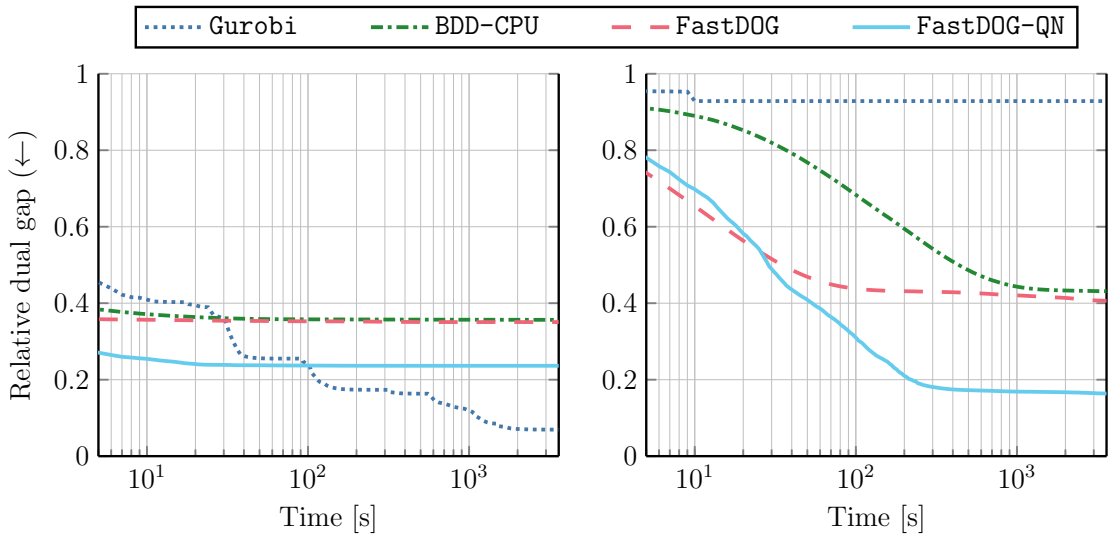
Figure 7.3: Convergence plots averaged over all instances of a dataset. The curves represent lower bounds (larger values are better) while markers denote objectives of rounded primal solutions (smaller values are better). The x-axis is plotted in log-scale.

the beginning better lower bounds than our baselines. For *QAPLib-small* dataset we additionally show convergence plot in Figure 7.4. In this case due to large variance in dual objectives within the dataset we compare solvers in terms of relative gaps g_t defined as

$$g_t = \min\left(\frac{E_{max} - E_t}{E_{max} - E_{min}}, 1\right), \tag{7.12}$$

where E_{max} denotes the largest (not necessarily optimal) lower bound to (BP), E_t the objective of relaxation (D) at time t and E_{min} is the minimum objective over all solvers (for normalization).

Discussion. In general, `FastDOG` gives up to an order of magnitude runtime improvement as compared to `BDD-CPU` from Lange and Swoboda (2021) and except on *worms* it achieves similar or better lower bounds. The variant `FastDOG-QN` sometimes lags behind `FastDOG` in early phase when inverse Hessian estimate is begin built. However later on `FastDOG-QN`



(a) *QAPLib-Small* (upto 50 vertices) (b) *QAPLib-Large* (56 to 128 vertices)

Figure 7.4: Relative dual gap (7.12) w.r.t. time on different splits of *QAPLib* dataset. The results are averaged on all instances within each split. The x-axis is plotted in log-scale.

achieves faster convergence and better dual objectives in almost all cases as compared to `FastDOG`. The small instances of *Graph matching* problems i.e., (*hotel*, *house*) are easily solved by `FastDOG` in less than 100 iterations therefore second order updates via `FastDOG-QN` do not yield further benefits.

In comparison to the respective hand-crafted `Specialized` CPU solvers our solvers `FastDOG` and `FastDOG-QN` are better for *Cell tracking* problems and worse for *Graph matching* problems (Fig. 7.3). On *MRF* problems both `FastDOG` and `FastDOG-QN` perform similar to the `Specialized` solver.

While `Gurobi` achieves, if given unlimited time, better lower bounds and primal solutions, our `FastDOG-QN` solver outperforms it on large instances under a one hour time limit. On small *QAPLib* instances `Gurobi` gives better lower bounds than all other solvers which get stuck in suboptimal fixed points (Fig. 7.4a).

In Figure 7.5 we compare per iteration performance of different methods for optimizing the Lagrangean relaxation (D) on *Shape matching* dataset. The parallel scheme in Alg. 7.1 needs roughly 5 times more iterations to reach the same lower bound as its sequential version in `BDD-CPU`. Hence, when counting the number of arithmetic operations the parallel scheme is less efficient in terms of objective improvement per iteration. Nonetheless, since more iterations of Alg. 7.1 can be performed per second this still leads to an overall faster algorithm. Lastly, we observe that `FastDOG-QN` outperforms both `FastDOG` and `BDD-CPU` as it additionally performs a quasi-Newton update in each iteration.

7.4.2 Limitations

As compared to `Gurobi` our schemes for both primal and dual optimization come without optimality guarantees. The dual optimization scheme is based on block coordinate ascent of non-smooth objective which can get stuck in suboptimal fixed points (Werner et al.,

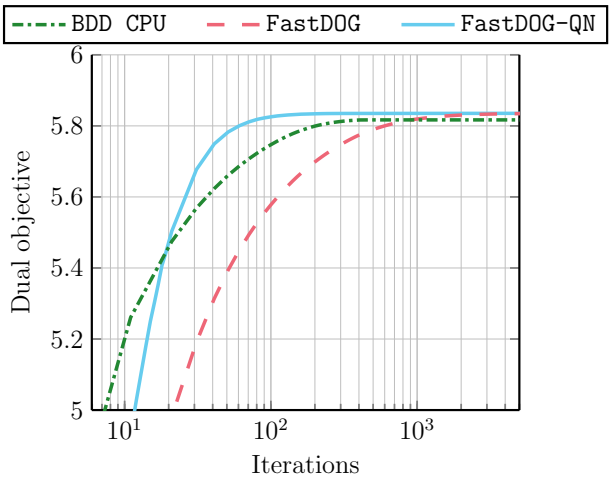


Figure 7.5: Comparison of schemes for solving the Lagrangean relaxation (D) w.r.t. number of iterations. The y-axis represents the lower bound (larger values are better) and x-axis denotes the number of iterations. Results are averaged over all instances of *Shape matching* dataset as also reported in Fig. 7.3d.

2020). Similarly for primal optimization the Algorithm 7.5 need not find a feasible solution for the NP-hard binary program (BP). However, empirically on the problem types we considered our primal heuristic did find good feasible solutions.

Technically our approach can also tackle problems with exponentially many inequalities such as the multicut problem (MC) via cutting planes. Our GPU implementation however, does not allow inserting additional inequalities efficiently. This is because reorganization of BDDs is required for maximum GPU utilization (see Figure 7.2).

7.5 Conclusion

We have proposed a massively parallelizable generic algorithm that can solve a wide variety of binary ILPs on GPU. Our results indicate that the performance of specialized efficient CPU solvers can be matched or even surpassed by a completely generic GPU solver. Our implementation is a first prototype and we conjecture that more speedups can be gained by elaborate implementation techniques e.g., compression of the BDD representation, better memory layout for better memory coalescing, multi-GPU support, etc. We argue that future improvements in optimization algorithms for structured prediction can be made by developing GPU friendly problem specific solvers and with improvements in our or other generic GPU solvers that can benefit many problem classes simultaneously. Another future avenue is optimization of ILPs from other domains e.g., on the MIPLib benchmark (Gleixner et al., 2021). These problems include integer and continuous-valued variable domains and constraints that are harder to represent as BDDs. Integer-valued variables can be tackled through multi-valued decision diagrams however handling continuous-valued variable remains a challenge. For handling exponential blowup in BDD representation additional encoding techniques are needed such as the works of Abío et al. (2012); Fujita et al. (2000) and Cappart et al. (2019).

8 Learning to Solve 0–1 ILP Relaxations

Contents

8.1 Introduction **97**
8.2 Related Work **99**
8.3 Method **100**
8.3.1 Lagrange Decomposition 100
8.3.2 Optimization of Lagrangean Dual 100
8.3.3 Backpropagation through Dual Optimization 101
8.3.4 Non-parametric Updates 104
8.3.5 Graph Neural Network 104
8.3.6 Overall Pipeline 106
8.4 Experiments **107**
8.4.1 Results 109
8.4.2 Limitations 112
8.5 Conclusion **112**

In this chapter we present a fast, scalable, data-driven approach for solving relaxations of 0-1 integer linear programs. We use a combination of graph neural networks (GNN) and the Lagrange decomposition based algorithm from Chapter 7. We generalize the latter, make it differentiable for end-to-end training, and use GNNs to predict its algorithmic parameters. This allows to retain the algorithm's theoretical properties including dual feasibility and guaranteed non-decrease in the lower bound while improving it via training. We overcome suboptimal fixed points of the basic solver by additional non-parametric GNN update steps maintaining dual feasibility. For training, we use a self-supervised loss. We train on smaller problems and test on larger ones showing strong generalization performance with a GNN comprising only around $10k$ parameters. Our solver achieves significantly faster performance and better dual objectives than its non-learned version, achieving close to optimal objective values of LP relaxations of very large structured prediction problems and on selected combinatorial ones. In particular, we achieve better objective values than specialized approximate solvers for specific problem classes while retaining their efficiency. Our solver has better any-time performance over a large time period compared to a commercial solver. Our implementation is available at https://github.com/LPMP/BDD.

8.1 Introduction

Integer linear programs (ILP) are a universal tool for solving combinatorial optimization problems. While great progress has been made on improving ILP solvers over the past

several decades, there is recent interest in leveraging machine learning to enhance ILP algorithms. Almost all ILP solving subroutines, except ILP relaxation algorithms, have been recently shown to benefit from learning, including variable selection for branch-and bound by Nair et al. (2020) or cutting plane selection (Huang et al., 2022; Turner et al., 2022; Paulus et al., 2022). Moreover a number of specialized heuristics as well as meta-algorithms using heuristics as subroutines have used ML for greatly improving performance for some problem classes (Sun and Yang, 2023; Qiu et al., 2022). However no general-purpose ILP relaxation algorithm has yet benefited from machine learning.

We make a contribution towards general ML-enabled solvers for optimization by learning a problem-agnostic solver for LP-relaxations of ILPs. LP solving is a key step taking most time in traditional ILP pipelines. State of the art LP solvers (Gurobi Optimization, 2019; CPLEX, 2019; FICO, 2022; MOSEK ApS, 2022; Gamrath et al., 2020) are not amenable to ML since they are non-differentiable, sequential and have very complex implementations. This makes utilization of neural networks and GPUs for solver improvement difficult. For these reasons we build upon the massively parallel solver from Chapter 7 and show that it can be made differentiable. This allows to train our problem agnostic solver for specific problem classes resulting in equal or better performance as compared to efficient hand-designed specialized solvers.

Contributions. Our high-level contributions are conceptual and empirical: (i) We show that embedding good inductive biases coming from non-learned solvers (i.e., the highly parallel block coordinate ascent algorithm from Chapter 7) into neural networks leads to greatly improved performance. In particular, we give evidence to the hypothesis that similar to vision (convolutions) and NLP (sequence models) the right inductive biases coming from solver primitives are a promising way to use the potential of ML for optimization. (ii) Our approach is more economical as compared to developing efficient problem specific heuristics, as is customary for large-scale problems in structured prediction tasks e.g., the works of Haller et al. (2020) for cell tracking and Hutschenreiter et al. (2021) for graph matching. Instead of spending much time and effort in designing and implementing new algorithms, one can train our problem agnostic solver with a few problem instances coming from the problem class of interest and obtain a state of the art GPU-enabled solver for it. To this end, we propose to learn the Lagrange decomposition algorithm of Chapter 7 for solving LP relaxations of ILPs and show its benefits. In particular,

- We generalize the deferred min-marginal averaging Algorithm 7.1 for optimizing the dual problem by allowing for a larger space of parameter updates.
- We make our dual optimization algorithm efficiently differentiable and embed it as a layer in a neural network. This enables us to predict parameters of the algorithm leading to faster convergence compared to manually designed rules.
- We train a predictor for arbitrary non-parametric updates that allow to escape suboptimal fixed points encountered by Alg. 7.1 (and our generalized variant).
- Our predictors for both of the above updates are trained in a fully self-supervised manner. Our loss optimizes for producing large improvements in the dual objective.
- We show the benefits of our learned approach on a wide range of problems. We have chosen large-scale structured prediction tasks of graph matching (Kainmueller et al.,

2014) and cell tracking (Haller et al., 2020). From theoretical computer science we compare on the QAPLib (Burkard et al., 1997) dataset and on randomly generated independent set problems (Prouvost et al., 2020).

8.2 Related Work

Learning to solve combinatorial optimization. ML has been used to improve various aspects of solving combinatorial problems. For the standard branch-and-cut ILP solvers the works by Gasse et al. (2019); Gupta et al. (2020); Nair et al. (2020); Gupta et al. (2022); Scavuzzo et al. (2022) learn variable selection for branching. The approaches by Ding et al. (2020); Nair et al. (2020) learn to fix a subset of integer variables in ILPs to their hopefully optimal values to improve finding high quality primal solutions. Variable selection for the large neighborhood search heuristic is explored by Sonnerat et al. (2021); Wu et al. (2021) for obtaining primal solutions to ILPs. Selecting good cuts by scoring them with neural networks was investigated by Huang et al. (2022); Turner et al. (2022). While all these approaches result in runtime and solution quality improvements, only a few works tackle the important task of speeding up ILP relaxations by ML. Specifically, graph neural network (GNN) were used by Cappart et al. (2019) to predict variable orderings of decision diagrams representing combinatorial optimization problems. The goal is to obtain an ordering such that a corresponding dual lower bound is maximal. To our knowledge it is the only work that accelerates ILP relaxation computation with ML. For constraint satisfaction problems, GNNs were used by Selsam et al. (2018); Cameron et al. (2020); Tönshoff et al. (2021) while the latter train in a self-supervised manner. For inference in graphical models parameters of belief propagation are predicted by neural networks for faster convergence by Deng et al. (2022), in a similar spirit to our work. However our method is applicable to a more general class of problems, allows escaping fixed points, and is scalable to larger problems due to efficient implementation. For narrow subclasses of problems primal heuristics have been augmented through learning some of their decisions e.g., for capacitated vehicle routing (Nazari et al., 2018), graph matching (Wang et al., 2021b) and traveling salesman (Xin et al., 2021). For a more complete overview of ML for combinatorial optimization we refer to the detailed surveys of Bengio et al. (2021) and Cappart et al. (2023).

Unrolling algorithms for parameter learning. Algorithms containing differentiable iterative procedures are combined with neural networks for improving performance of such algorithms. Such approaches show more generalization power than pure neural networks based ones as shown in the survey of Monga et al. (2021). The work of Gregor and LeCun (2010) embedded sparse coding algorithms in a neural network by unrolling. For solving inverse problems the works of Yang et al. (2020b); Chen and Pock (2017) unroll through ADMM and non-linear diffusion resp. Lastly, neural networks were used to predict update directions for training other neural networks e.g., by Andrychowicz et al. (2016) and Metz et al. (2022).

8.3 Method

We give an overview of the Lagrange decomposition approach to binary ILPs and generalize the dual optimization scheme (Alg. 7.1) for faster convergence. Then we will show how to backpropagate through the optimization scheme allowing to train a graph neural network for predicting its parameters. For more detailed introduction we refer to Sec. 6.1.

8.3.1 Lagrange Decomposition

Recall the binary program (BP)

$$\min_{x\in\{0,1\}^{\mathcal{I}}} \langle c, x\rangle \quad \text{s.t.} \quad x_{\mathcal{I}_j} \in \mathcal{X}_j \quad \forall j \in \mathcal{J}\,, \tag{BP}$$

and its Lagrangean relaxation (D)

$$\max_{\lambda} \quad \sum_{j\in\mathcal{J}} \min_{x\in\mathcal{X}_j} \langle \lambda_j, x\rangle \quad \text{s.t.} \quad \sum_{j\in\mathcal{J}_i} \lambda_{ij} = c_i \quad \forall i \in \mathcal{I}\,. \tag{D}$$

The problem (D) provides a lower bound to the NP-hard optimization problem (BP) and is also useful for primal recovery e.g., via Alg. 7.5. Our goal is to learn a neural network for optimizing the dual (D) efficiently and to reach better objective values.

8.3.2 Optimization of Lagrangean Dual

In chapter 7 we proposed a parallelization friendly iterative scheme for optimizing (D) with hand-designed parameters. We generalize this scheme in Algorithm 8.1, exposing a much larger set of parameters allowing more control over the optimization process. Since this large parameter space is difficult to be tuned manually, we will employ a GNN for predicting these parameters.

In detail, Alg. 8.1 greedily assigns the Lagrange variables in u-many disjoint blocks $B_1, \dots, B_u$ in such a way that each block contains at most one Lagrange variable from each subproblem and all variables within a block are updated in parallel (same as Alg. 7.1). As before, the dual update scheme relies on computing min-marginal differences. These min-marginal differences are averaged out across subproblems via updates to Lagrange variables. Our algorithm relies on two important set of parameters, damping factors and averaging weights which are set separately for each Lagrange variable. The damping factor ω_{ij} determines the fraction of min-marginal difference to subtract from variable i in subproblem j. The averaging weight α_{ij} parameterizes the fraction of total min-marginal difference ($\sum_{ik} M_{ik}$) variable i in subproblem j receives. Note that we have described Alg. 8.1 in a slightly different form than Alg. 7.1. This is to allow for an easier description of its backpropagation routine in the forthcoming section.

Remark. *The deferred min-marginal averaging algorithm Alg. 7.1 is a specialized form of our generalized Algorithm 8.1 if the parameters were set as* $\omega_{ij} = 0.5$ *and* $\alpha_{ij} = \frac{1}{|\mathcal{J}_i|}$ *for all* i, j.

Algorithm 8.1: Generalized Min-Marginal Averaging (`GenMMA`)

Input: Lagrange variables $\lambda_{ij}\,\forall i \in \mathcal{I}, j \in \mathcal{J}_i$,
damping factors $\omega_{ij} \in (0,1)\,\forall i \in \mathcal{I}, j \in \mathcal{J}_i$,
averaging weights $\alpha_{ij} \in (0,1)\,\forall i \in \mathcal{I}, j \in \mathcal{J}_i$,
max. number of iterations T.

1 Initialize deferred min-marginal diff. $M = 0$
2 **for** T *iterations* **do**
3 **for** *block* $B \in (B_1, \dots B_u)$ **do**
4 $\lambda, M \leftarrow$ `BlockUpdate` $(B, \lambda, M, \alpha, \omega)$
5 **end**
6 **for** *block* $B \in (B_u, \dots B_1)$ **do**
7 $\lambda, M \leftarrow$ `BlockUpdate` $(B, \lambda, M, \alpha, \omega)$
8 **end**
9 **end**
10 **return** λ, M
11 **Procedure** `BlockUpdate`$(B, \lambda^{\texttt{in}}, M^{\texttt{in}}, \alpha, \omega)$
12 **for** $ij \in B$ *in parallel* **do**
 `// Compute min-marginal diff. and scale by damping factor`
13 $M_{ij}^{\texttt{out}} = \omega_{ij} \cdot \Big[\min_{\substack{x \in \mathcal{X}_j: \\ x_i = 1}} \langle \lambda_j^{\texttt{in}}, x \rangle - \min_{\substack{x \in \mathcal{X}_j: \\ x_i = 0}} \langle \lambda_j^{\texttt{in}}, x \rangle \Big]$
 `// Subtract current min-marginal diff. and distribute previous one w.r.t. averaging weights`
14 $\lambda_{ij}^{\texttt{out}} = \lambda_{ij}^{\texttt{in}} - M_{ij}^{\texttt{out}} + \alpha_{ij} \cdot \sum_{k \in \mathcal{J}_i} M_{ik}^{\texttt{in}}$
15 **end**
16 **return** $\lambda^{\texttt{out}}$, $M^{\texttt{out}}$

We generalize the min-marginal update step by considering damping factors ω_{ij} in $[0,1]$ and averaging weights to be arbitrary convex combinations ($\alpha_{ij} \geq 0$ with $\sum_{j \in \mathcal{J}_i} \alpha_{ij} = 1$). This generalized update step still preserves the desirable property of guaranteed non-improvement in the dual objective.

Proposition 19 (Dual Feasibility and Monotonicity of Generalized Min-marginal Averaging)**.** *For any $\alpha_{ij} \geq 0$ with $\sum_{j \in \mathcal{J}_i} \alpha_{ij} = 1$ and $\omega_{ij} \in [0,1]$ the min-marginal averaging step in line 14 in Algorithm 8.1 retains dual feasibility and is non-decreasing in the dual lower bound.*

Proof. Can be seen by setting α and ω appropriately in Prop. 18 from the previous chapter. □

8.3.3 Backpropagation through Dual Optimization

We show below how to differentiate through Algorithm 8.1 with respect to its parameters α and ω. This will ultimately allow us to learn these parameters such that faster convergence is achieved. To this end, we describe backpropagation for a block update (lines 11-16) of

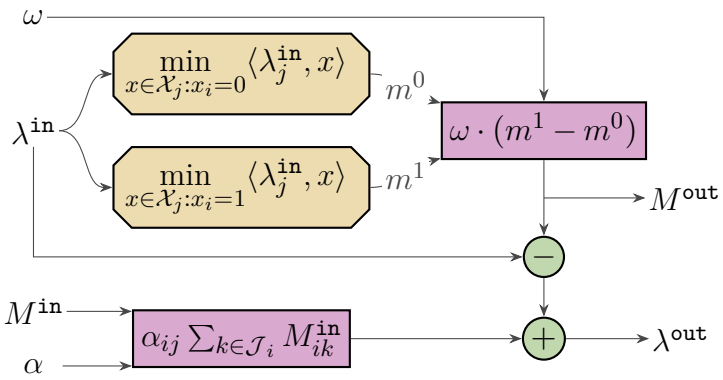


Figure 8.1: Computational graph of `BlockUpdate` from Alg. 8.1

Alg. 8.1. All other operations can be tackled by automatic differentiation. For a block B in $\{B_1,\dots,B_u\}$ we view the Lagrangean update as a mapping $\mathcal{H}:(\mathbb{R}^{|B|})^4\to(\mathbb{R}^{|B|})^2$, $(\lambda^{\mathtt{in}},M^{\mathtt{in}},\alpha,\omega)\mapsto(\lambda^{\mathtt{out}},M^{\mathtt{out}})$.

Given a loss function $\mathcal{L}:\mathbb{R}^N\to\mathbb{R}$ we denote $\partial\mathcal{L}/\partial x$ by $\dot{x}$. Algorithm 8.2 shows backpropagation through $\mathcal{H}$ to compute the gradients $\dot{\lambda}^{\mathtt{in}}$, $\dot{M}^{\mathtt{in}}$, $\dot{\alpha}$ and $\dot{\omega}$.

Algorithm 8.2: `BlockUpdate` backpropagation

Input: Forward pass inputs: $B,\lambda^{\mathtt{in}},M^{\mathtt{in}},\alpha,\omega$, gradients of forward pass output: $\dot{\lambda}^{\mathtt{out}}$, $\dot{M}^{\mathtt{out}}$, gradients of parameters $\dot{\alpha},\dot{\omega}$

for $ij\in B$ *in parallel* **do**
- $\dot{M}_{ij}^{\mathtt{in}}=\sum_{k\in\mathcal{J}_i}\dot{\lambda}_{ik}^{\mathtt{out}}\alpha_{ik}$
- $\dot{M}_{ij}^{\mathtt{out}}=\dot{M}_{ij}^{\mathtt{out}}-\dot{\lambda}_{ij}^{\mathtt{out}}$
- $\dot{\alpha}_{ij}=\dot{\alpha}_{ij}+\dot{\lambda}_{ij}\sum_{k\in\mathcal{J}_i}M_{ik}^{\mathtt{in}}$
- $\dot{\omega}_{ij}=\dot{\omega}_{ij}+\dot{M}_{ij}^{\mathtt{out}}[M_{ij}^{\mathtt{out}}/\omega_{ij}]$
- Compute for $\beta\in\{0,1\}$ $s^j(i,\beta)=\operatorname{arg\,min}_{x\in\mathcal{X}_j:x_i=\beta}\langle\lambda_j^{\mathtt{in}},x\rangle$
- $\dot{\lambda}_{pj}^{\mathtt{in}}=\dot{\lambda}_{pj}^{\mathtt{out}}+\dot{M}_{ij}^{\mathtt{out}}\omega_{ij}[s_p^j(i,1)-s_p^j(i,0)]$, $\forall p\in\mathcal{I}_j$

end

return $\dot{\lambda}^{\mathtt{in}},\dot{M}^{\mathtt{in}},\dot{\alpha},\dot{\omega}$

Proposition 20. *Alg. 8.2 performs backpropagation through* $\mathcal{H}$.

Proof. The computational graph of `BlockUpdate` in Alg. 8.1 is shown in Figure 8.1. Assuming gradients $\partial\mathcal{L}/\partial M^{\mathtt{out}}$ and $\partial\mathcal{L}/\partial\lambda^{\mathtt{out}}$ are given. We first focus on lower part of Figure 8.1. By applying chain rule gradient of $M_{ij}^{\mathtt{in}}\,\forall ij\in B$ is computed as

$$\frac{\partial\mathcal{L}}{\partial M_{ij}^{\mathtt{in}}}=\sum_{p\in\mathcal{I}}\sum_{k\in\mathcal{J}_p}\frac{\partial\mathcal{L}}{\partial\lambda_{pk}^{\mathtt{out}}}\frac{\partial\lambda_{pk}^{\mathtt{out}}}{\partial M_{ij}^{\mathtt{in}}}=\sum_{k\in\mathcal{J}_i}\frac{\partial\mathcal{L}}{\partial\lambda_{ik}^{\mathtt{out}}}\frac{\partial\lambda_{ik}^{\mathtt{out}}}{\partial M_{ij}^{\mathtt{in}}}=\sum_{k\in\mathcal{J}_i}\frac{\partial\mathcal{L}}{\partial\lambda_{ik}^{\mathtt{out}}}\alpha_{ij}. \tag{8.1}$$

Similarly gradient for $\alpha_{ij}\,\forall ij\in B$ is

$$\frac{\partial\mathcal{L}}{\partial\alpha_{ij}}=\sum_{p\in\mathcal{I}}\sum_{k\in\mathcal{J}_p}\frac{\partial\mathcal{L}}{\partial\lambda_{pk}^{\mathtt{out}}}\frac{\partial\lambda_{pk}^{\mathtt{out}}}{\partial\alpha_{ij}}=\frac{\partial\mathcal{L}}{\partial\lambda_{ij}^{\mathtt{out}}}\frac{\partial\lambda_{ij}^{\mathtt{out}}}{\partial\alpha_{ij}}=\frac{\partial\mathcal{L}}{\partial\lambda_{ij}^{\mathtt{out}}}\sum_{k\in\mathcal{J}_i}M_{ik}^{\mathtt{in}}, \tag{8.2}$$

Since we allow running Alg. 8.1 for more than one iteration with same parameters (α, ω), the above gradient (8.2) is accumulated to existing gradients of α to obtain the result given by Alg. 8.2.

For the upper part of Figure 8.1 we first backpropagate gradients of $\lambda^{\mathtt{out}}$ to $M^{\mathtt{out}}$ to account for subtraction $(-)$ as

$$\frac{\partial \mathcal{L}}{\partial M^{\mathtt{out}}} = \frac{\partial \mathcal{L}}{\partial M^{\mathtt{out}}} - \frac{\partial \mathcal{L}}{\partial \lambda^{\mathtt{out}}}. \tag{8.3}$$

Then the gradient w.r.t. damping factors $\omega_{ij}\, \forall ij \in B$ is

$$\frac{\partial \mathcal{L}}{\partial \omega_{ij}} = \frac{\partial \mathcal{L}}{\partial M^{\mathtt{out}}_{ij}} \frac{\partial M^{\mathtt{out}}_{ij}}{\partial \omega_{ij}} = \frac{\partial \mathcal{L}}{\partial M^{\mathtt{out}}_{ij}} \left(m^1_{ij} - m^0_{ij} \right) = \frac{\partial \mathcal{L}}{\partial M^{\mathtt{out}}_{ij}} \left(\frac{M^{\mathtt{out}}_{ij}}{\omega_{ij}} \right), \tag{8.4}$$

which also needs to be accumulated to existing gradient as done for gradients of α.

Lastly, to backpropagate gradients to $\lambda^{\mathtt{in}}$ we first calculate

$$\frac{\partial \mathcal{L}}{\partial m^0_{ij}} = \frac{\partial \mathcal{L}}{\partial M^{\mathtt{out}}_{ij}} \frac{\partial M^{\mathtt{out}}_{ij}}{\partial m^0_{ij}} = -\frac{\partial \mathcal{L}}{\partial M^{\mathtt{out}}_{ij}} \omega_{ij}, \tag{8.5a}$$

$$\frac{\partial \mathcal{L}}{\partial m^1_{ij}} = \frac{\partial \mathcal{L}}{\partial M^{\mathtt{out}}_{ij}} \frac{\partial M^{\mathtt{out}}_{ij}}{\partial m^1_{ij}} = \frac{\partial \mathcal{L}}{\partial M^{\mathtt{out}}_{ij}} \omega_{ij}. \tag{8.5b}$$

Then (sub-)gradient of min-marginals $m^0_{ij}, m^1_{ij}\, \forall ij \in B$ w.r.t. $\lambda^{\mathtt{in}}$ is

$$\frac{\partial m^\beta_{ij}}{\partial \lambda^{\mathtt{in}}_j} = \operatorname*{arg\,min}_{x \in \mathcal{X}_j : x_{ij} = \beta} \langle \lambda^{\mathtt{in}}_j, x \rangle, \quad \forall \beta \in \{0, 1\}. \tag{8.6}$$

Using the above relations (8.5), (8.6) and applying chain rule we obtain

$$\frac{\partial \mathcal{L}}{\partial \lambda^{\mathtt{in}}_{ij}} = \frac{\partial \mathcal{L}}{\partial \lambda^{\mathtt{out}}_{ij}} + \sum_{\beta \in \{0,1\}} \sum_{p \in \mathcal{I}} \sum_{k \in \mathcal{J}_p} \frac{\partial \mathcal{L}}{\partial m^\beta_{pk}} \frac{\partial m^\beta_{pk}}{\lambda^{\mathtt{in}}_{ij}} \tag{8.7a}$$

$$= \frac{\partial \mathcal{L}}{\partial \lambda^{\mathtt{out}}_{ij}} + \sum_{\beta \in \{0,1\}} \sum_{p \in \mathcal{I}_j} \frac{\partial \mathcal{L}}{\partial m^\beta_{pj}} \frac{\partial m^\beta_{pj}}{\lambda^{\mathtt{in}}_{ij}}, \; \forall ij \in B. \tag{8.7b}$$

□

Efficient implementation. Generally, the naive computation of min-marginal differences and its backpropagation are both expensive operations as they require solving two optimization problems for each dual variable. Similar to Chapter 7 here we also represent each subproblem by a binary decision diagram (BDD) for fast computation of min-marginal differences. The final algorithm results in a computation graph involving only elementary arithmetic operations and taking minima over several variables. Using this computational graph we can implement the abstract Algorithm 8.2 efficiently on GPU. For further performance gains we implement custom routines for backpropagation.

8.3.4 Non-parametric Updates

Although the min-marginal averaging scheme of Alg. 8.1 guarantees a non-decreasing lower bound, it can get stuck in suboptimal fixed points. A discussion for the special case of MAP inference in Markov Random Fields is by Werner (2007) and a more general setting by Werner et al. (2020). To address this issue we allow arbitrary updates to Lagrange variables through a vector $\hat{\theta} \in \mathbb{R}^{|\lambda|}$ as

$$\lambda_{ij} \leftarrow \lambda_{ij} + \hat{\theta}_{ij} - \frac{1}{|\mathcal{J}_i|} \sum_{k \in \mathcal{J}_i} \hat{\theta}_{ik} \quad \forall i \in \mathcal{I}, j \in \mathcal{J}_i, \tag{8.8}$$

where the last term ensures feasibility of updated Lagrange variables w.r.t. the dual problem (D).

8.3.5 Graph Neural Network

A graph neural network (GNN) is used to predict the parameters $\alpha, \omega \in \mathbb{R}^{|\lambda|}$ of Alg. 8.1 and also the non-parametric update $\theta \in \mathbb{R}^{|\lambda|}$ for (8.8). To this end, the dual problem (D) is encoded on a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The nodes $\mathcal{V}$ correspond to primal variables $\mathcal{I}$ and subproblems $\mathcal{J}$ i.e., $\mathcal{V} = \mathcal{I} \cup \mathcal{J}$ and edges $\mathcal{E} = \{ij \mid i \in \mathcal{I}, j \in \mathcal{J}_i\}$ correspond to Lagrange multipliers. We need to predict values of α_{ij}, ω_{ij} and θ_{ij} for each edge ij in $\mathcal{E}$.

Hand-crafted features. We associate features $f = (f_{\mathcal{I}}, f_{\mathcal{J}}, f_{\mathcal{E}})$ with each entity (nodes, edges) of the bipartite graph. Lagrange multipliers $\lambda^{\texttt{in}}$ and deferred min-marginals $M^{\texttt{in}}$ encode the current state of Alg. 8.1 as a part of edge features. Additionally, we provide a number of quantities which can allow the GNN to make better updates. Moving average of previously computed features is provided for information about the goodness of past updates, thus allowing for change of (implicit) step-sizes. A subgradient of the dual problem (D) is encoded in the edge features $f_{\mathcal{E}}$. This enables the GNN to effectively utilize more information in parameter prediction than conventional hand-designed updates rules can manage. For example min-marginal averaging schemes can get stuck in suboptimal fixed points due to zero min-marginal differences (Werner et al., 2020). Since the GNN additionally has access to subgradient of the dual problem it can escape such fixed points. In addition we provide gradients of smoothed dual objective (D) as edge features. To this end, we replace each subproblem $E^j(\lambda_j)$ with its smoothed variant (Lange and Swoboda, 2021, Sec. A.5)

$$E^j_\alpha(\lambda_j) = \alpha \cdot \log \left(\sum_{x \in \mathcal{X}_j} \exp\left(\frac{\langle \lambda_j, x \rangle}{\alpha} \right) \right), \tag{8.9}$$

with varying values of smoothing factor $\alpha > 0$. A complete list of features is provided in Table 8.1

Graph convolution. We use the transformer based graph convolution scheme (Shi et al., 2021). As a first step, embeddings of all subproblems j in $\mathcal{J}$ are computed by receiving messages from adjacent nodes and edges as

$$\texttt{CONV}_{\mathcal{J}}(f_{\mathcal{I}}, f_{\mathcal{J}}, f_{\mathcal{E}}, \mathcal{E})_j = \mathbf{W_s} f_j + \sum_{i:ij \in \mathcal{E}} a_{ij}(f_j, f_{\mathcal{I}}, f_{\mathcal{E}}; \mathbf{W_a}) \left[\mathbf{W_t} f_i + \mathbf{W_e} f_{ij} \right], \tag{8.10}$$

Table 8.1: List of hand-crafted features provided to the GNN. Exponentially averaged features are computed with a smoothing factor of 0.9. Features corresponding to the ILP remain fixed (i.e., node degrees, constraint type, c, A, b) whereas the remaining features are updated based on current Lagrange variables λ.

Types	Feature description
Primal variables $f_{\mathcal{I}}$	Normalized cost vector $c/\|c\|_\infty$ Node degree ($\|\mathcal{J}_i\| \forall i \in \mathcal{I}$)
Subproblems $f_{\mathcal{J}}$	Node degree ($\|\mathcal{I}_j\| \forall j \in \mathcal{J}$) RHS vector b in constraints $Ax \leq b$ Indicator for constraint type ($\leq$ or $=$) Current objective value per subproblem $[E^1(\lambda_1), \ldots, E^m(\lambda_j)]$ Exp. moving avg. of first, second order change in objective Change in objective value from last non-parametric update (8.8)
Dual variables $f_{\mathcal{E}}$	Normalized Lagrange variables Normalized deferred min-marginal differences Coefficients of constraint matrix A Optimal assignment of each subproblem (i.e., subgradient of (D)) Exp. moving average of the optimal assignment Gradient of the smoothed objective (8.9) for all α in $\{1.0, 10.0, 100.0\}$

where $\mathbf{W_a}, \mathbf{W_s}, \mathbf{W_t}, \mathbf{W_e}$ are trainable parameters and $a_{ij}(f_j, f_{\mathcal{I}}, f_{\mathcal{E}}; \mathbf{W_a})$ is the softmax attention weight between nodes i and j parameterized by $\mathbf{W_a}$. The above step is repeated in the reverse direction to compute embeddings for variables $\mathcal{I}$. A similar strategy for message passing on bipartite graphs was done by Gasse et al. (2019).

Recurrent connections. Our default GNN as mentioned above only uses hand-crafted features to maintain a history of previous optimization rounds. To learn a summary of the past updates we optionally allow recurrent connections through an LSTM with forget gate (Gers et al., 1999). The LSTM is only applied on primal variable nodes $\mathcal{I}$ and maintains cell states $s_{\mathcal{I}}$ which can be updated and used for parameter prediction in subsequent optimization rounds.

Prediction. The learned embeddings from GNN, LSTM outputs and solver features are consumed by a multi-layer perceptron Φ to predict the required variables for each edge ij in $\mathcal{E}$. Afterwards we transform these outputs so that they satisfy Prop. 19. The exact sequence of operations performed by the graph neural network are shown in Alg. 8.3 where $[u_1, \ldots, u_k]$ denotes concatenation of vectors $u_1, \ldots, u_k$, `LN` denotes layer normalization (Ba et al., 2016) and $\texttt{LSTM}_{\mathcal{I}}$ stands for an LSTM cell operating on primal variables $\mathcal{I}$.

Loss. Given the Lagrange variables λ we directly use the dual objective (D) as a self-supervised loss to train the GNN. Thus, we maximize the loss L defined as

$$\mathcal{L}(\lambda) = \sum_{j \in \mathcal{J}} \min_{x \in \mathcal{X}_j} \langle \lambda_j, x \rangle \tag{8.11}$$

For a mini-batch of instances during training we take the mean of corresponding per-instance losses. For backpropagation, gradient of the loss $\mathcal{L}$ w.r.t. Lagrange variables of

Algorithm 8.3: Parameter prediction by GNN

Input: Primal variable features $f_{\mathcal{I}}$ and cell states $s_{\mathcal{I}}$, Subproblem features $f_{\mathcal{J}}$, Dual variable (edge) features $f_{\mathcal{E}}$, Set of edges $\mathcal{E}$.

$h_{\mathcal{J}} = \texttt{ReLU}\left(\texttt{LN}\left(\texttt{CONV}_{\mathcal{J}}\left(f_{\mathcal{I}}, f_{\mathcal{J}}, f_{\mathcal{E}}, \mathcal{E}\right)\right)\right)$ // Compute subproblems embeddings

$h_{\mathcal{I}} = \texttt{ReLU}\left(\texttt{LN}\left(\texttt{CONV}_{\mathcal{I}}\left(f_{\mathcal{I}}, [f_{\mathcal{J}}, h_{\mathcal{J}}], f_{\mathcal{E}}, \mathcal{E}\right)\right)\right)$ // Compute primal var embeddings

$z_{\mathcal{I}}, s_{\mathcal{I}} = \texttt{LSTM}_{\mathcal{I}}(h_{\mathcal{I}}, s_{\mathcal{I}})$ // Compute output and cell state

$(\hat{\alpha}, \hat{\omega}, \hat{\theta}) = \Phi\left([f_{\mathcal{I}}, h_{\mathcal{I}}, z_{\mathcal{I}}], [f_{\mathcal{J}}, h_{\mathcal{J}}], f_{\mathcal{E}}, \mathcal{E}\right)$ // Prediction per edge

$[\alpha_{ij}]_{j \in \mathcal{J}_i} = \texttt{Softmax}\left([\hat{\alpha}_{ij}]_{j \in \mathcal{J}_i}\right), \quad \forall i \in \mathcal{I},$
$\omega_{ij} = \texttt{Sigmoid}(\hat{\omega}_{ij}), \quad \forall i \in \mathcal{I}, j \in \mathcal{J}_i$ // Ensure non-decreasing obj., (Prop 19)

$\theta_{ij} = \hat{\theta}_{ij} - \frac{1}{|\mathcal{J}_i|} \sum_{k \in \mathcal{J}_i} \hat{\theta}_{ik}, \quad \forall i \in \mathcal{I}, j \in \mathcal{J}_i$ // Maintain dual feasibility

return $\alpha, \omega, \theta, s_{\mathcal{I}}$

a subproblem j is computed by finding a minimizing assignment for that subproblem, written as

$$\left(\frac{\partial \mathcal{L}}{\partial \lambda}\right)_j = \arg\min_{x \in \mathcal{X}_j} \langle \lambda_j, x \rangle \in \{0,1\}^{\mathcal{I}_j}. \tag{8.12}$$

This gradient is then sent as input for backpropagation. For computing the minimizing assignment efficiently we use binary decision diagram representation of each subproblem.

8.3.6 Overall Pipeline

We train our pipeline (Fig. 8.2) which contains multiple dual optimization rounds in a fashion similar to that of recurrent neural networks. One round of our dual optimization consists of message passing by GNN, a non-parametric update step and T iterations of generalized min-marginal averaging. For computational efficiency we run our pipeline for at most R dual optimization rounds during training. On each mini-batch we randomly sample a number of optimization rounds r in $[R]$, run $r-1$ rounds without tracking gradients and backpropagate through the last round by computing the loss (8.11). For the pipeline with recurrent connections we backpropagate through last 3 rounds and apply the loss after each of these rounds. Since the task of dual optimization is relatively easier in early rounds as compared to later ones we use two neural networks. The early stage network is trained if the randomly sampled r is in $[0, R/2]$ and the late stage network is chosen otherwise. During testing we switch to the later stage network when the relative improvement in the dual objective by the early stage network becomes less than 10^{-6}. For computational efficiency during testing we query the GNN for parameter updates only after $T \gg 1$ iterations of Alg. 8.1.

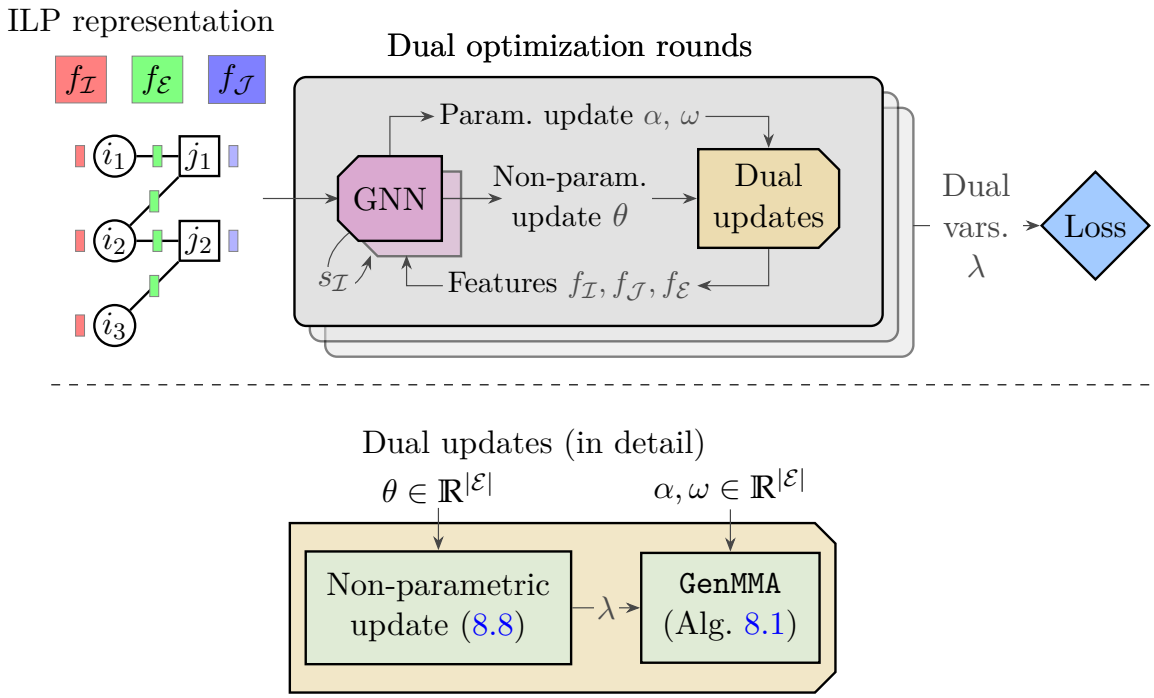


Figure 8.2: (Top:) Our pipeline for optimizing the Lagrangean dual (D). The dual problem is encoded on a bipartite graph containing features $f_{\mathcal{I}}$, $f_{\mathcal{J}}$ and $f_{\mathcal{E}}$ for primal variables, subproblems and dual variables resp. A graph neural network (GNN) predicts θ, α, ω for dual updates. In one dual update block (bottom), current set of Lagrange multipliers λ are first updated by the non-parametric update using θ and parametric update is done via Alg. 8.1 using α, ω. The updated solver features f and LSTM cell states $s_{\mathcal{I}}$ are sent to the GNN in next optimization round. These rounds are repeated at most R-times during training and until convergence of the dual objective (D) during inference.

8.4 Experiments

Evaluation. As main evaluation metric we report convergence plots of the relative dual gap $g(t) \in [0, 1]$ at time t by

$$g(t) = \min\left(\frac{d^* - d(t)}{d^* - d_{init}}, 1.0\right) \tag{8.13}$$

where $d(t)$ is the dual objective at time t, d^* is the optimal (or best known) objective value of the Lagrange relaxation (D) and d_{init} is the objective before optimization as also computed in Sec 7.4. Additionally we also report per dataset averages of the best objective value (E) and time taken (t) to obtain the best objective.

Datasets. We evaluate our approach on a variety of datasets from different domains. For each dataset we train our pipeline on smaller instances and test on larger ones.

Cell tracking: Instances of developing flywing tissue from cell tracking challenge (Ulman et al., 2017) processed by Haller et al. (2020) and made available by Swoboda et al. (2022b). We use the largest and hardest 3 instances, train on the 2 smaller instances and test on the largest one.

Graph matching: Instances of graph matching for matching nuclei in 3D microscopic images (Long et al., 2009) processed by Kainmueller et al. (2014) and made publicly available as ILPs by Swoboda et al. (2022b). We train on 10 instances and test on the remaining 20.

Independent set: Random instances of independent set problem generated by the library of Prouvost et al. (2020). For training we use 240 instances with $10k$ nodes each and test on 60 instances with $50k$ nodes.

QAPLib: The benchmark dataset for quadratic assignment problems used in the combinatorial optimization community (Burkard et al., 1997). The benchmark contains problems arising from a variety of domains e.g., keyboard design, hospital layout, circuit synthesis, facility location, etc. We train on 61 instances having up to 30 nodes and test on 35 instances containing up to 70 nodes. Conversion to ILP is done via (2.7)-(2.16) of Loiola et al. (2007)

For each dataset the size of dual problems (D) are reported in Table 8.2.

Table 8.2: Statistics of datasets where the values are averaged within each train/test split. Number of edges in the GNN equal the number of Lagrange multipliers λ.

Dataset	# variables ($\times 10^6$)		# constraints ($\times 10^6$)		# edges ($\times 10^6$)	
	train	test	train	test	train	test
Cell tracking	3.1	10.1	0.6	2.2	8.5	27.5
Graph matching	1.5	0.1	1.5	0.1	3.3	3.3
Independent set	0.01	0.05	0.04	0.4	0.1	1.1
QAPLib	0.1	2.5	0.02	0.2	0.6	10.6

Hyperparameters. Due to varying instance sizes we use a separate set of hyperparameters for each dataset. The hyperparameters used in experiments are reported in Table 8.3. During training time we run the Alg. 8.1 for only a few iterations for computational efficiency since more iterations can make the backward pass much slower due to reverse mode autodifferentiation. For test time we run Alg. 8.1 for more iterations since backward pass is not required. For *QAPLib* dataset we need more training time than other datasets because training set is quite large and has more diversity as compared to other datasets. Hyperparameter validation is done on the training set itself.

For the *cell tracking* dataset we only predict $\theta \in \mathbb{R}^{|\lambda|}$ for non-parametric update steps (8.8) and fix the parameters α, ω in Alg. 8.1 to their default values from Chapter 7. Learning these parameters gave worse performance on the training set following our evaluation protocol. We attribute this to lack of generalization to longer time-horizons (Table 8.3).

Algorithms.

`Gurobi`: The dual simplex algorithm from the commercial solver (Gurobi Optimization, 2019). For *QAPLib* dataset we run both dual simplex and interior point methods and choose the best performing one for each instance.

Table 8.3: Hyperparameters of our approach for each dataset. T: Num. of iterations of Alg. 8.1 in each optimization round; R: max. number of training rounds; # itr. train: Num. of training iterations.

Dataset	T		R	batch size	learn. rate	# itr. train	train time [hrs]
	train	test					
Cell tracking	1	100	400	1	1e-3	500	14
Graph matching	20	200	20	2	1e-3	400	4
Independent set	20	50	20	8	1e-3	2500	10
QAPLib	5	20	500	4	1e-3	1600	48

`Spec.`: For *graph matching* and *cell tracking* datasets we also report results of state-of-the-art dataset specific solvers. For *cell tracking* the solver of Haller et al. (2020) and for *graph matching* the best performing solver (fm-bca) from recent benchmark of Haller et al. (2022).

`FastDOG`: The non-learned baseline via Alg. 7.1 with hand-designed parameters $\omega_{ij} = 0.5$ and $\alpha_{ij} = 1/|\mathcal{J}_i|$ as a specialization of Alg. 8.1.

`FastDOG-QN`: The variant of `FastDOG` with additional quasi-Newton updates (Alg. 7.4). From the experiments in Sec. 7.4 it was the best performing general-purpose algorithm on large instances.

`DOGE`: Our approach where we learn to predict parametric and non-parametric updates by using two graph neural networks for early and late-stage optimization. Size of the learned embeddings h computed by the GNN in Alg. 8.3 is set to 16 for nodes and 8 for edges. For computing attention weights in (8.10) we use only one attention head for efficiency. The predictor Φ in Alg. 8.3 contains 4 linear layers with the ReLU activation. We train the networks using the Adam optimizer (Kingma and Ba, 2014). To prevent gradient overflow we use gradient clipping on model parameters by an l^2 norm of 50. The number of trainable parameters is $8k$.

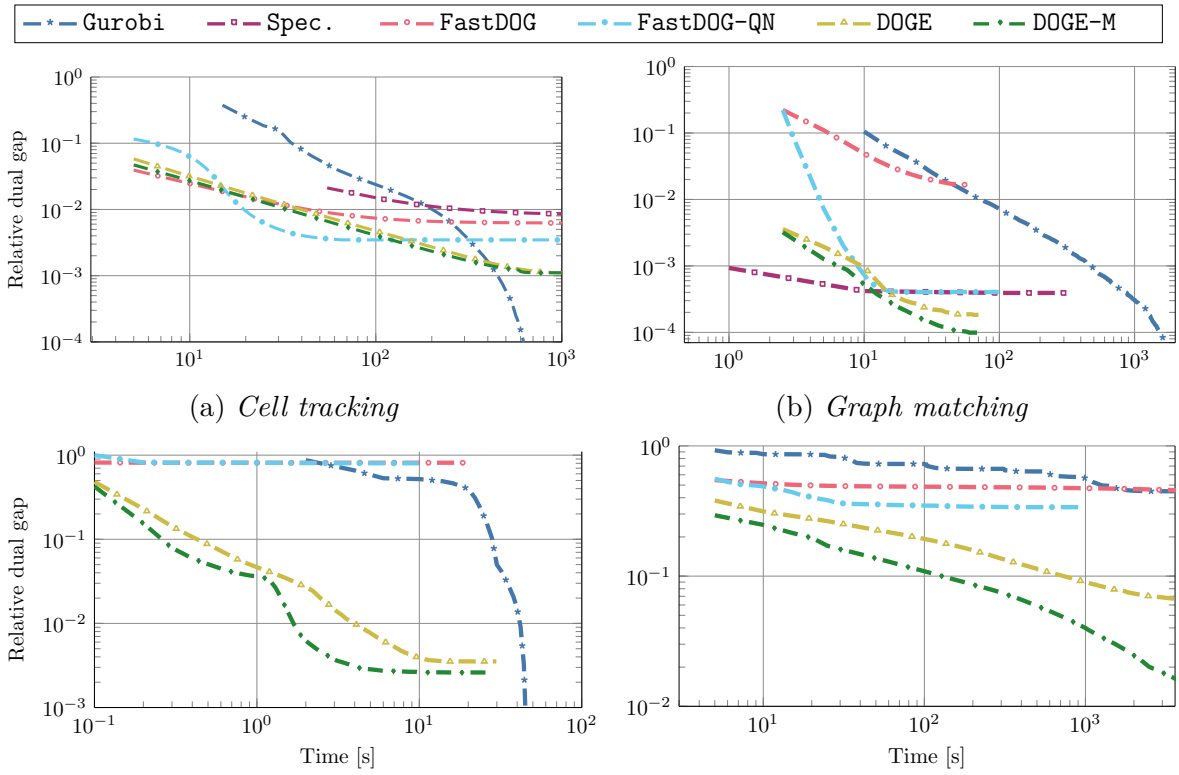
`DOGE-M`: Variant of our method where we additionally use recurrent connections using LSTM. The cell state vector s_i for each primal variable node $i \in \mathcal{I}$ has a size of 16. The number of trainable parameters is $12k$.

Note the test instances require millions of solver parameters to be predicted (ref. Table 8.2) while our largest GNN has $12k$ parameters.

For training we use PyTorch and implement the Algorithms 8.1 and 8.2 in CUDA. CPU solvers use AMD EPYC 7702 CPU with 16 threads. GPU solvers use either an NVIDIA RTX 8000 (48GB) or a A100 (80GB) GPU depending on problem size.

8.4.1 Results

For each dataset we evaluate our methods on corresponding testing split. Convergence plots of relative dual gaps change (averaged over all test instances) are given in Figure 8.3. Other evaluation metrics are reported in Table 8.4.



(a) *Cell tracking* (b) *Graph matching*

(c) *Independent set* (d) *QAPLib*

Figure 8.3: Convergence plots for $g(t)$ (8.13) the relative dual gap to the optimum (or maximum suboptimal objective among all methods) of the relaxation (D). X-axis indicates wall clock time and both axes are logarithmic. The value of $g(t)$ is averaged over all test instances in each dataset.

Table 8.4: Results on test instances where the values are averaged within a dataset. Numbers in bold highlight the best performance and underlines indicate the second best objective. E: objective value (D) (larger values are better), $t[s]$: runtime (in seconds) to attain objective E.

	Cell tracking		*Graph matching*		*Independent set*		*QAPLib*	
	$E(\times 10^8)$	$t[s]$	$E(\times 10^4)$	$t[s]$	$E(\times 10^4)$	$t[s]$	$E(\times 10^6)$	$t[s]$
`Gurobi`	**−3.852**	809	**−4.8433**	278	**−2.4457**	52	9.5	3600
`Spec.`	−3.866	1673	−4.8443	100	-	-	-	-
`FastDOG`	−3.863	1005	−4.8912	61	−2.4913	9	5.7	1680
`FastDOG-QN`	−3.858	75	−4.8444	10	−2.4907	7	7.8	246
`DOGE`	−3.854	1015	−4.8439	17	−2.4460	8	<u>12.1</u>	720
`DOGE-M`	<u>−3.854</u>	730	<u>−4.8436</u>	21	<u>−2.4459</u>	5	**14.5**	861

Discussion. As compared to the non-learned hand-designed solvers `FastDOG` and `FastDOG-QN` we reach much more accurate relaxation solutions, almost closing the gap to optimum as computed by `Gurobi`. Even though given unlimited time `Gurobi` attains the optimum, we often reach reasonably close values that are considered correct for practical purposes. For example the graph matching benchmark (Haller et al., 2022) considers a relative gap less than 10^{-3} as optimal (we achieve 10^{-4}). Moreover our learned solvers reach much better objective values as compared to specialized solvers. Using LSTM in `DOGE-M` further improves the performance especially on the most difficult *QAPLib* dataset. On *QAPLib* `Gurobi` does not converge on instances with more than 40 nodes within the time limit of one hour. Our difference to `Gurobi` is most pronounced w.r.t. anytime performance as our solver reaches good solutions relatively early.

Ablation study. We evaluate the importance of various components in our approach. Starting from Alg. 7.1 (`FastDOG`) as a baseline we first predict all parameters α, ω, θ through the two multi-layer perceptrons Φ for early and late stage optimization without using the GNN. Next, we report results of using one network (instead of two) which is trained and tested for both early and later rounds of dual optimization. Lastly, we aim to seek the importance of learning parameters of Alg. 8.2 and the non-parametric update (8.8). To this end, we learn to predict only the non-parametric update and apply the loss directly on updated λ without requiring backpropagation through Alg. 8.1. We also try learning a subset of parameters i.e., not predicting averaging weights α or damping factors ω. Lastly, we report results of `DOGE-M` which uses recurrent connections. The results for *graph matching* dataset are in Table 8.5.

Table 8.5: Ablation study results on the *Graph matching* dataset. w/o GNN: Use only the two predictors Φ without GNN for early and late stage optimization; same network: use one network (GNN, Φ) for both early and late stage; only non-param., param.: predict only the non-parametric update (8.8) or the parametric update (Alg. 8.1); w/o α, ω: does not predict α or ω resp.

	`FastDOG`	w/o GNN	same network	only non-param.	only param.	w/o α	w/o ω	`DOGE`	`DOGE-M`
E ($\uparrow$)	-48912	-48440	-48444	-48476	-48444	-48439	-48439	-48439	$\mathbf{-48436}$
$t[s]$ ($\downarrow$)	61	29	24	51	74	30	30	17	21

Firstly, from our ablation study we observe that learning even one of the two types of updates i.e., non-parametric or parametric already gives better results than the non-learned solver `FastDOG`. This is because non-parametric update can help in escaping fixed points when they occur and the parametric update can help Alg. 8.1 in avoiding such fixed points. Combining both of these strategies further improves the results. Secondly, we observe that GNN gives improvement over only using the MLP. Thirdly, we find using separate networks for early and late stage optimization gives better performance than using the same network for all stages. Lastly, using recurrent connections through an LSTM gives the best performance.

CUDA implementation. The Algorithm 8.1 can be implemented via scatter, gather operations available in Pytorch (Paszke et al., 2019) and Pytorch Geometric (Fey and

Lenssen, 2019) making use of automatic differentiation and alleviating the need for custom backpropagation routines. This however leads to high GPU memory requirements for large instances and also hinders training of the GNN. Therefore for further computational efficiency we implement Algorithm 8.1 and its backpropagation in CUDA (NVIDIA et al., 2021; Hoberock and Bell, 2010) and expose via pybind (Jakob et al., 2017). Since backpropagation requires intermediate forward pass iterates, this strategy allows to trade memory by (re-)computation during backpropagation of Alg. 8.1. A comparison between our Pytorch implementation which relies on automatic differentation with our CUDA implementation is given in Table 8.6. We observe that for all datasets containing large instances (i.e., all datasets except *Independent set*) GPU memory usage is drastically reduced through our CUDA implementation. Additionally, runtimes for both forward and backward pass are reduced.

Table 8.6: Runtime and peak GPU memory usage statistics of one generalized min-marginal averaging iteration in Algorithm 8.1 and its backpropagation via Algorithm 8.2. FP, BP: Forward pass (one iteration of Alg. 8.1) and its backward pass resp.; mem.: Maximum GPU memory in GB used during both forward and backward pass. The values are averaged over all training instances within each dataset.

	Cell tracking			*Graph matching*			*Independent set*			*QAPLib*		
	time [ms]		mem. (GB)	time [ms]		mem. (GB)	time [ms]		mem. (GB)	time [ms]		mem. (GB)
	FP	BP		FP	BP		FP	BP		FP	BP	
PyTorch	305	1039	31	153	344	11	11	16	0.7	43.5	70	7.3
CUDA	16	171	3.4	7	68	1.8	0.7	5.7	0.7	1.6	16	1.6

8.4.2 Limitations

Easy problem classes, including small *cell tracking* (Haller et al., 2020) and easy Markov Random Field (MRF) inference (Kappes et al., 2013) do not benefit from learning, since `FastDOG` already solves the problem in few iterations. Some problem classes have sequential bottlenecks due to long subproblems, including MRFs for protein folding (Jaimovich et al., 2006) and shape matching (Windheuser et al., 2011a,b), which makes training difficult due to slow dual optimization.

Although our method requires training for each problem class, the cost of training is manageable. Nonetheless devising a generalizable approach is an interesting research direction requiring at least: a large and diverse training set, powerful neural network and multi-GPU implementation.

8.5 Conclusion

We have proposed an self-supervised learning approach for solving relaxations to combinatorial optimization problems by backpropagating through and learning parameters for

an ILP relaxation solver. We demonstrated its potential in obtaining close to optimal solutions much faster than with traditional methods. Although our solvers require training as compared to conventional solvers, this overhead is negligible as compared to the human effort required for developing efficient specialized solvers (which are also often outperformed by our approach).

Our work generalizes efficient approximate solver development: instead of developing a specialized solver we propose to use a generically applicable one and train it to obtain fast and accurate optimization algorithm. Going one step further and training a universal model that generalizes across different problem classes remains a challenge for future work.

9 Conclusion

This thesis studied methods for large-scale combinatorial optimization (CO) problems arising from structured output prediction tasks in visual computing.

In the first part we studied a particular class of CO problems i.e., the multicut problem and its variants, for applications in partitioning tasks from computer vision. We studied modeling aspects of the multicut problem i.e., specifying graph structure and edge costs. For predicting the edge costs we utilized neural networks in an end-to-end fashion by backpropagating through the multicut problem. Our approach resulted in a better segmentation quality as compared to existing methods on comparable neural network architectures. For the problem of graph structure design we circumvented the problem by devising a compact formulation (i.e., dense multicut) on fully connected graphs accompanied by efficient algorithms. The resulting method yielded better panoptic segmentation as compared to multicut on a hand-designed graph. Lastly, to overcome the scalability issues of multicut algorithms on large problem sizes and meet real-time performance requirements we developed massively parallel algorithms achieving a speed-up of more than an order of magnitude over the prior art.

Our study of the multicut problem opens interesting future research directions. Since our work studied three aspects of the multicut problem separately, considering them in unison is also appealing. For example, our dense multicut formulation offers greater expressiveness than the conventional one and thus can benefit neural network pipelines for clustering problems. However more work needs to be done for devising its backpropagation scheme. Another interesting topic is devising massively parallel algorithms for variants of the multicut problem such as asymmetric multiway cut, lifted multicut (Keuper et al., 2015), and our proposed dense multicut case. Going one step further, development of a unified solver along with backpropagation mechanisms for such approaches can be a long term goal in this direction.

In the second part we aimed towards a general-purpose and efficient integer programming (IP) approach for tackling CO problems for structured prediction tasks. For improving performance of CO algorithms we sought a means to exploit machine learning and parallel computing. To this end we forewent conventional IP paradigms and studied parallelizable and differentiable solver primitives. We first devised parallel schemes for solving IP relaxations via parallel dual block coordinate ascent and primal decoding by cost perturbation. The resulting GPU algorithm offered better anytime performance as compared to a general-purpose commercial solver and an on-par performance with domain-specific sequential solvers. In the last chapter we utilized machine learning (ML) via graph neural networks for solving IP relaxations. Our learning enhanced solver yielded substantial improvement over its non-learned counterpart on many problem classes and achieved better solutions as compared to two hand-designed domain-specific CPU solvers.

Our work is a first prototype in the direction of speeding-up relaxation solvers through

ML. Our learned solver requires training for each problem class and does not generalize across problem classes. Progress in this direction might require a more powerful neural network, diverse dataset, better training scheme, multi-GPU implementation etc. A related work in this direction is from Metz et al. (2022) where ML is used for optimizing other neural networks. Another avenue is making our cost perturbation based primal decoding scheme more powerful e.g., by backtracking, maintaining history, etc. Utilizing ML for the latter is an interesting open problem.

Although, our CO algorithms work reasonably well for many structured prediction problems and a few cases from operations research, an even wider applicability requires further investigation. Specifically, our schemes for optimizing the Lagrangean relaxation cannot provably find the optimal solution and our primal heuristics also lack guarantees. While the latter limitation can be side-stepped by employing for example a branch-and-bound framework, more work need to be done to overcome the former. Specifically, an open problem is to make our relaxation algorithms provably optimal while retaining their desirable traits of being differentiable and parallelizable.

In general terms we investigated a longstanding question of how to effectively leverage CO in ML and ML in CO. For the former we improved techniques for gradient estimation through CO problems yielding better panoptic segmentation results than comparable approaches. For the latter we showed a possibility by designing a solver containing differentiable primitives while also being general-purpose. To enable both of these aspects we devised massively parallel algorithms with efficient GPU implementations yielding benefits (among others) in faster training of neural networks containing such CO solvers.

List of Algorithms

2.1 Greedy Additive Edge Contraction (GAEC) . . . 21

3.1 Backward pass through panoptic segmentation . . . 33

4.1 Parallel Edge Contraction (PEC) . . . 44
4.2 Parallel Message Passing (PMP) . . . 46
4.3 Primal-Dual Multicut . . . 47
4.4 Parallel Edge Contraction pseudocode . . . 48
4.5 Parallel Conflicted 5-Cycles pseudocode . . . 49

5.1 DenseGAEC . . . 58
5.2 Incremental NN update . . . 61

7.1 Parallel Deferred Min-Marginal Averaging (ParallelMMA) . . . 86
7.2 Forward Pass Min-Marginal Computation . . . 88
7.3 Backward Pass Min-Marginal Computation . . . 88
7.4 Quasi-Newton update with Min-Marginal Averaging . . . 89
7.5 Perturbation Primal Rounding . . . 91

8.1 Generalized Min-Marginal Averaging (GenMMA) . . . 101
8.2 BlockUpdate backpropagation . . . 102
8.3 Parameter prediction by GNN . . . 106

List of Figures

2.1 Multicut of a graph . . . 19
3.1 Overview of architecture for panoptic segmentation . . . 28
3.2 Example multiway cut and asymmetric multiway cut problem solutions . . . 30
3.3 Gradient computation through asymmetric multiway cut . . . 33
3.4 Training convergence comparison of our robust gradient estimation technique 37
3.5 Panoptic quality surrogate versus exact panoptic quality . . . 37
4.1 Edge contraction example . . . 43
4.2 Example iteration of our primal-dual multicut solver . . . 47
4.3 Comparison of multicut upper bounds . . . 51
4.4 Comparison of multicut lower bounds . . . 52
4.5 Comparison of runtime over different problem sizes . . . 52
5.1 Example illustration of dense multicut problem . . . 57
5.2 Example illustration of nearest neighbor search after edge contraction . . . 60
5.3 Example illustration of dense multicut in $2D$ space . . . 62
5.4 ImageNet runtime comparison . . . 67
6.1 Example of a binary program with its Lagrangean decomposition . . . 73
6.2 Illustration of a weighted Binary Decision Diagram (BDD) . . . 77
7.1 Example decomposition of a binary program . . . 85
7.2 BDD nodes arrangement in GPU memory . . . 88
7.3 Convergence plots on all datasets . . . 94
7.4 Convergence plots for small and large *QAPLib* instances . . . 95
7.5 Comparison of algorithms in terms of number of iterations . . . 96
8.1 Computational graph of `GenMMA` . . . 102
8.2 Pipeline for learning to solve ILP relaxations . . . 107
8.3 Lower bound convergence plots for all datasets . . . 110

List of Tables

Tab. 3.1 Panoptic segmentation results . . . 36
Tab. 3.2 Comparison of PQ surrogate loss with separate losses on AMWC output 38
Tab. 4.1 Comparison of lower and upper bounds on all datasets . . . 51
Tab. 4.2 Runtime breakdown . . . 52
Tab. 5.1 Quality of clustering on ImageNet validation set . . . 65
Tab. 5.2 Comparison of algorithms for dense multicut problem on ImageNet . 66
Tab. 5.3 Comparison of panoptic segmentation on sparse and dense graph . . . 67
Tab. 5.4 Comparison of algorithms for dense multicut problem on Cityscapes . 68
Tab. 5.5 Influence of affinity strength . . . 68
Tab. 6.1 Notation for Lagrangean decomposition of a binary program . . . 72
Tab. 7.1 Comparison of lower bounds, upper bounds, and runtimes on all datasets 93
Tab. 8.1 List of hand-crafted features for GNN . . . 105
Tab. 8.2 Dataset statistics . . . 108
Tab. 8.3 Hyperparameters . . . 109
Tab. 8.4 Lower bounds comparison on all datasets . . . 110
Tab. 8.5 Ablation study results . . . 111
Tab. 8.6 Efficiency comparison of custom CUDA implementation . . . 112

Bibliography

A. Abbas and P. Swoboda. Combinatorial optimization for panoptic segmentation: A fully differentiable approach. *Advances in Neural Information Processing Systems*, 34:15635–15649, 2021. Cited on pages 14, 65, 66, and 67.

A. Abbas and P. Swoboda. FastDOG: Fast discrete optimization on GPU. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022a. Cited on pages 15, 77, and 78.

A. Abbas and P. Swoboda. RAMA: A Rapid Multicut Algorithm on GPU. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8193–8202, June 2022b. Cited on pages 14, 63, and 83.

A. Abbas and P. Swoboda. ClusterFuG: clustering fully connected graphs by multicut. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023. Cited on page 14.

A. Abbas and P. Swoboda. Doge-train: Discrete optimization on gpu with end-to-end training. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(18):20623–20631, Mar. 2024. doi: 10.1609/aaai.v38i18.30048. URL https://ojs.aaai.org/index.php/AAAI/article/view/30048. Cited on page 15.

I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and V. Mayer-Eichberger. A new look at bdds for pseudo-boolean constraints. *Journal of Artificial Intelligence Research*, 45: 443–480, 2012. Cited on page 96.

W. P. Adams. Improved linear programming based lower bounds for the quadratic assignment problem. *Quadratic assignment and related problems, DIMACS series in discrete mathematics and theoretical computer science*, 16:43–77, 1994. Cited on page 71.

A. Agrawal, S. Barratt, and S. Boyd. Learning convex optimization models. *arXiv preprint arXiv:2006.04248*, 2020. Cited on page 27.

T. Ajanthan, A. Desmaison, R. Bunel, M. Salzmann, P. H. Torr, and M. Pawan Kumar. Efficient linear programming for dense crfs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3298–3306, 2017. Cited on page 74.

A. Alush and J. Goldberger. Ensemble segmentation using efficient integer linear programming. *IEEE transactions on pattern analysis and machine intelligence*, 34(10):1966–1977, 2012. Cited on page 22.

A. Alush and J. Goldberger. Break and conquer: Efficient correlation clustering for image segmentation. In *International Workshop on Similarity-Based Pattern Recognition*, pages 134–147. Springer, 2013. Cited on page 23.

B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017. Cited on page 27.

H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A constraint store based on multivalued decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 118–132. Springer, 2007. Cited on page 84.

B. Andres, J. H. Kappes, T. Beier, U. Köthe, and F. A. Hamprecht. Probabilistic image segmentation with closedness constraints. In *ICCV*, 2011. doi: 10.1109/ICCV.2011.6126550. Cited on page 23.

B. Andres, T. Kröger, K. L. Briggman, W. Denk, N. Korogod, G. Knott, U. Köthe, and F. A. Hamprecht. Globally optimal closed-surface segmentation for connectomics. In *ECCV*, 2012. doi: 10.1007/978-3-642-33712-3_56. Cited on page 23.

B. Andres, J. Yarkony, B. Manjunath, S. Kirchhoff, E. Turetken, C. C. Fowlkes, and H. Pfister. Segmenting planar superpixel adjacency graphs wrt non-planar superpixel affinity graphs. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 266–279. Springer, 2013. Cited on page 23.

M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016. Cited on page 99.

D. Applegate, M. Díaz, O. Hinder, H. Lu, M. Lubin, B. O'Donoghue, and W. Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. In *NeurIPS 2021*, 2021. Cited on page 74.

I. Arganda-Carreras, S. C. Turaga, D. R. Berger, D. Cireşan, A. Giusti, L. M. Gambardella, J. Schmidhuber, D. Laptev, S. Dwivedi, J. M. Buhmann, T. Liu, M. Seyedhosseini, T. Tasdizen, L. Kamentsky, R. Burget, V. Uher, X. Tan, C. Sun, T. D. Pham, E. Bas, M. G. Uzunbas, A. Cardona, J. Schindelin, and H. S. Seung. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9:142, 2015. ISSN 1662-5129. doi: 10.3389/fnana.2015.00142. Cited on page 38.

A. Arnab, S. Zheng, S. Jayasumana, B. Romera-Paredes, M. Larsson, A. Kirillov, B. Savchynskyy, C. Rother, F. Kahl, and P. H. Torr. Conditional random fields meet deep neural networks for semantic segmentation: Combining probabilistic graphical models with deep learning for structured prediction. *IEEE Signal Processing Magazine*, 35(1):37–52, 2018. Cited on page 27.

D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 9780898716245. Cited on page 64.

Y. Asano, C. Rupprecht, and A. Vedaldi. Self-labelling via simultaneous clustering and representation learning. In *International Conference on Learning Representations*, 2019. Cited on page 79.

B. F. Auer and R. H. Bisseling. Graph coarsening and clustering on the GPU. *Graph Partitioning and Graph Clustering*, 588:223, 2012. Cited on page 42.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. Cited on page 105.

S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019. Cited on page 27.

A. Bailoni, C. Pape, S. Wolf, T. Beier, A. Kreshuk, and F. A. Hamprecht. A generalized framework for agglomerative clustering of signed graphs applied to instance segmentation. *arXiv preprint arXiv:1906.11713*, 2019. Cited on page 41.

A. Bailoni, C. Pape, N. Hütsch, S. Wolf, T. Beier, A. Kreshuk, and F. A. Hamprecht. GASP, a Generalized Framework for Agglomerative Clustering of Signed Graphs and Its Application to Instance Segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11645–11655, 2022. Cited on page 21.

D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981. Cited on page 26.

N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine learning*, 56(1-3):89–113, 2004. Cited on pages 19, 20, and 40.

O. Bastert, B. Hummel, and S. de Vries. A generalized wedelin heuristic for integer programming. *INFORMS Journal on Computing*, 22(1):93–107, 2010. Cited on page 90.

T. Beier, T. Kroeger, J. H. Kappes, U. Kothe, and F. A. Hamprecht. Cut, glue & cut: A fast, approximate solver for multicut partitioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014. Cited on pages 22 and 40.

T. Beier, B. Andres, U. Köthe, and F. A. Hamprecht. An efficient fusion move algorithm for the minimum cost lifted multicut problem. In *European Conference on Computer Vision*. Springer, 2016. Cited on pages 22 and 40.

T. Beier, C. Pape, N. Rahaman, T. Prange, S. Berg, D. D. Bock, A. Cardona, G. W. Knott, S. M. Plaza, L. K. Scheffer, et al. Multicut brings automated neurite segmentation closer to human performance. *Nature methods*, 14(2):101, 2017. Cited on page 23.

Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. Cited on page 31.

Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. Cited on page 99.

D. Bergman and A. A. Cire. Decomposition based on decision diagrams. In C.-G. Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 45–54, Cham, 2016. Springer International Publishing. Cited on page 83.

D. Bergman and A. A. Cire. Discrete nonlinear optimization by state-space decompositions. *Management Science*, 64(10):4700–4720, 2018. Cited on page 83.

D. Bergman, A. A. Cire, and W.-J. van Hoeve. Lagrangian bounds from decision diagrams. *Constraints*, 20(3):346–361, 2015. Cited on page 84.

D. Bergman, A. A. Cire, W.-J. Van Hoeve, and J. Hooker. *Decision diagrams for optimization*, volume 1. Springer, 2016a. Cited on page 84.

D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016b. Cited on page 84.

M. Berman, A. R. Triki, and M. B. Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4413–4421, 2018. Cited on page 38.

G. Bertasius, Q. Liu, L. Torresani, and J. Shi. Local perturb-and-map for structured prediction. In *Artificial Intelligence and Statistics*, pages 585–594. PMLR, 2017. Cited on page 27.

Q. Berthet, M. Blondel, O. Teboul, M. Cuturi, J.-P. Vert, and F. Bach. Learning with differentiable perturbed optimizers. *arXiv preprint arXiv:2002.08676*, 2020. Cited on pages 25 and 28.

Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001. Cited on page 93.

G. Brasó and L. Leal-Taixé. Learning a neural solver for multiple object tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6247–6257, 2020. Cited on pages 79 and 80.

R. E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986. Cited on pages 77 and 78.

R. Bunel, A. De Palma, A. Desmaison, K. Dvijotham, P. Kohli, P. Torr, and M. P. Kumar. Lagrangian decomposition for neural network verification. In *Conference on Uncertainty in Artificial Intelligence*, pages 370–379. PMLR, 2020. Cited on page 74.

R. E. Burkard, S. E. Karisch, and F. Rendl. QAPLIB–a quadratic assignment problem library. *Journal of Global optimization*, 10(4):391–403, 1997. Cited on pages 92, 99, and 108.

G. Calinescu. *Multiway Cut*, pages 567–569. Springer US, Boston, MA, 2008. ISBN 978-0-387-30162-4. doi: 10.1007/978-0-387-30162-4_253. Cited on page 29.

C. Cameron, R. Chen, J. Hartford, and K. Leyton-Brown. Predicting Propositional Satisfiability via End-to-End Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34 (04):3324–3331, Apr. 2020. doi: 10.1609/aaai.v34i04.5733. Cited on page 99.

Q. Cappart, E. Goutierre, D. Bergman, and L.-M. Rousseau. Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1443–1451, 2019. Cited on pages 96 and 99.

Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Velickovic. Combinatorial optimization and reasoning with graph neural networks. *J. Mach. Learn. Res.*, 24:130–1, 2023. Cited on page 99.

N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020. Cited on pages 26 and 27.

M. P. Castro, A. A. Cire, and J. C. Beck. An mdd-based lagrangian approach to the multi-commodity pickup-and-delivery tsp. *INFORMS Journal on Computing*, 32(2):263–278, 2020. Cited on page 84.

O. Cetintas, G. Brasó, and L. Leal-Taixé. Unifying short and long-term tracking with graph hierarchies. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22877–22887, 2023. Cited on page 79.

L.-C. Chen, A. G. Schwing, A. L. Yuille, and R. Urtasun. Learning deep structured models, 2015. Cited on page 27.

L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018. Cited on pages 25 and 29.

X. Chen, S. Xie, and K. He. An empirical study of training self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9640–9649, 2021. Cited on page 64.

Y. Chen and T. Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1256–1272, 2016. Cited on page 27.

Y. Chen and T. Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1256–1272, 2017. doi: 10.1109/TPAMI.2016.2596743. Cited on page 99.

B. Cheng, M. D. Collins, Y. Zhu, T. Liu, T. S. Huang, H. Adam, and L.-C. Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12475–12485, 2020. Cited on pages 25, 26, 29, 32, 34, 35, 36, and 79.

S. Chopra and M. R. Rao. On the multiway cut polyhedron. *Networks*, 21(1):51–89, 1991. Cited on page 23.

S. Chopra and M. R. Rao. The partition problem. *Mathematical Programming*, 59(1-3):87–115, 1993. Cited on pages 19, 20, 45, and 54.

B. Christianson. Reverse accumulation and attractive fixed points. *Optimization Methods and Software*, 3(4):311–326, 1994. Cited on page 27.

J. Cohen and P. Castonguay. Efficient graph matching and coloring on the gpu. In *GTC*. NVIDIA, 2012. Cited on page 44.

M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016. Cited on pages 34, 41, 49, 63, and 65.

C. Corro and I. Titov. Differentiable Perturb-and-Parse: Semi-Supervised Parsing with a Structured Variational Autoencoder. In *International Conference on Learning Representations*, 2019. Cited on page 28.

CPLEX. CPLEX Optimization Studio 12.10, 2019. Cited on pages 15, 82, and 98.

M. Cuturi, O. Teboul, and J.-P. Vert. Differentiable ranking and sorting using optimal transport. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, 2019. Cited on page 27.

E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2-3):172–187, 2006. Cited on pages 19, 20, and 40.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. Cited on pages 29, 63, and 64.

Y. Deng, S. Kong, C. Liu, and B. An. Deep attentive belief propagation: Integrating reasoning and learning for solving constraint optimization problems. *Advances in Neural Information Processing Systems*, 35:25436–25449, 2022. Cited on page 99.

M. Deza, M. Grötschel, and M. Laurent. Clique-web facets for multicut polytopes. *Mathematics of Operations Research*, 17(4):981–1000, 1992. Cited on page 20.

I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007. doi: 10.1109/TPAMI.2007.1115. Cited on page 56.

J.-Y. Ding, C. Zhang, L. Shen, S. Li, B. Wang, Y. Xu, and L. Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1452–1459, 2020. Cited on page 99.

J. Djolonga and A. Krause. Differentiable learning of submodular models. *Advances in Neural Information Processing Systems*, 30:1013–1023, 2017. Cited on page 27.

J. Domke. Implicit differentiation by perturbation. *Advances in Neural Information Processing Systems*, 23:523–531, 2010. Cited on pages 25, 27, 28, and 31.

J. Domke. Learning graphical model parameters with approximate marginal inference. *IEEE transactions on pattern analysis and machine intelligence*, 35(10):2454–2467, 2013. Cited on page 27.

A. Ferber, B. Wilder, B. Dilkina, and M. Tambe. Mipaal: Mixed integer program as a layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1504–1511, 2020. Cited on page 25.

M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. Cited on page 111.

FICO. FICO Xpress Optimization Suite, 2022. Cited on page 98.

M. Fujita, Y. Lu, E. Clarke, and J. Jain. Efficient Variable Ordering using aBDD based Sampling. In *Design Automation Conference*, pages 687–692, Los Alamitos, CA, USA, jun 2000. IEEE Computer Society. doi: 10.1109/DAC.2000.855402. Cited on page 96.

J. Funke, S. Saalfeld, D. Bock, S. Turaga, and E. Perlman. CREMI MICCAI Challenge on circuit reconstruction from Electron Microscopy Images, 2016. URL https://cremi.org. Cited on page 49.

G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. L. Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig. The SCIP Optimization Suite 7.0. Technical Report 20-10, ZIB, Takustr. 7, 14195 Berlin, 2020. Cited on page 98.

N. Gao, Y. Shan, Y. Wang, X. Zhao, Y. Yu, M. Yang, and K. Huang. SSAP: Single-Shot Instance Segmentation With Affinity Pyramid . In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 642–651, 2019. Cited on pages 25, 26, 35, and 36.

M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*, 2019. Cited on pages 83, 99, and 105.

Z. Geng, C. Wang, Y. Wei, Z. Liu, H. Li, and H. Hu. Human pose as compositional tokens. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 660–671, 2023. Cited on page 80.

F. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with LSTM. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pages 850–855 vol.2, 1999. doi: 10.1049/cp:19991218. Cited on page 105.

A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. M. Christophel, K. Jarck, T. Koch, J. Linderoth, M. Lübbecke, H. D. Mittelmann, D. Ozyurt, T. K. Ralphs, D. Salvagnin, and Y. Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021. doi: 10.1007/s12532-020-00194-3. Cited on page 96.

A. Globerson and T. S. Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *Advances in neural information processing systems*, pages 553–560, 2008. Cited on page 74.

J. Gondzio and R. Sarkissian. Parallel interior-point solver for structured linear programs. *Mathematical Programming*, 96(3):561–584, 2003. Cited on page 83.

J. E. González, A. A. Cire, A. Lodi, and L.-M. Rousseau. BDD-based optimization for the quadratic stable set problem. *Discrete Optimization*, page 100610, 2020. Cited on page 84.

J. E. González, A. A. Cire, A. Lodi, and L.-M. Rousseau. Integrated integer programming and decision diagram search tree with an application to the maximum independent set problem. *Constraints*, pages 1–24, 2020. Cited on page 84.

S. Gould, R. Hartley, and D. Campbell. Deep declarative networks: A new hope. *arXiv preprint arXiv:1909.04866*, 2019. Cited on page 27.

K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 399–406, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077. Cited on page 99.

M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45:59–96, 1989. Cited on page 20.

M. Grötschel and Y. Wakabayashi. Facets of the clique partitioning polytope. *Mathematical Programming*, 47(1):367–387, 1990. Cited on page 20.

M. Guignard and S. Kim. Lagrangean decomposition: A model yielding stronger lagrangean bounds. *Mathematical programming*, 39(2):215–228, 1987. Cited on page 72.

P. Gupta, M. Gasse, E. Khalil, P. Mudigonda, A. Lodi, and Y. Bengio. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33:18087–18097, 2020. Cited on page 99.

P. Gupta, E. B. Khalil, D. Chetélat, M. Gasse, Y. Bengio, A. Lodi, and M. P. Kumar. Lookback for learning to branch. *arXiv preprint arXiv:2206.14987*, 2022. Cited on page 99.

L. Gurobi Optimization. Gurobi optimizer reference manual, 2019. URL http://www.gurobi.com. Cited on pages 15, 22, 82, 91, 92, 98, and 108.

S. Haller, M. Prakash, L. Hutschenreiter, T. Pietzsch, C. Rother, F. Jug, P. Swoboda, and B. Savchynskyy. A primal-dual solver for large-scale tracking-by-assignment. In *AISTATS*, 2020. Cited on pages 72, 74, 92, 98, 99, 107, 109, and 112.

S. Haller, L. Feineis, L. Hutschenreiter, F. Bernard, C. Rother, D. Kainmüller, P. Swoboda, and B. Savchynskyy. A comparative study of graph matching algorithms in computer vision. In *Proceedings of the European Conference on Computer Vision*, 2022. Cited on pages 71, 74, 92, 109, and 111.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a. Cited on page 64.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016b. Cited on page 28.

K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. Cited on page 26.

J. Hoberock and N. Bell. Thrust: A parallel template library, 2010. URL http://thrust.github.io/. Version 1.7.0. Cited on pages 48, 49, 92, and 112.

J. N. Hooker. Improved job sequencing bounds from decision diagrams. In T. Schiex and S. de Givry, editors, *Principles and Practice of Constraint Programming*, pages 268–283, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30048-7. Cited on page 84.

A. Hornakova, R. Henschel, B. Rosenhahn, and P. Swoboda. Lifted disjoint paths with application in multiple object tracking. In *International Conference on Machine Learning*, pages 4364–4375. PMLR, 2020. Cited on page 23.

A. Hornakova, T. Kaiser, P. Swoboda, M. Rolinek, B. Rosenhahn, and R. Henschel. Making higher order MOT scalable: An efficient approximate solver for lifted disjoint paths. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6330–6340, 2021. Cited on pages 72 and 74.

T. C. Hu. Multi-commodity network flows. *Operations research*, 11(3):344–360, 1963. Cited on page 20.

Z. Huang, K. Wang, F. Liu, H.-L. Zhen, W. Zhang, M. Yuan, J. Hao, Y. Yu, and J. Wang. Learning to select cuts for efficient mixed-integer programming. *Pattern Recognition*, 123: 108353, 2022. Cited on pages 98 and 99.

Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Math. Program. Comput.*, 10(1):119–142, 2018. doi: 10.1007/s12532-017-0130-5. Cited on page 83.

L. Hutschenreiter, S. Haller, L. Feineis, C. Rother, D. Kainmüller, and B. Savchynskyy. Fusion moves for graph matching. In *ICCV*, 2021. Cited on pages 72 and 98.

H. C. Indelman and T. Hazan. Learning randomly perturbed structured predictors for direct loss minimization. *arXiv preprint arXiv:2007.05724*, 2020. Cited on pages 25 and 31.

E. Insafutdinov, M. Andriluka, L. Pishchulin, S. Tang, E. Levinkov, B. Andres, and B. Schiele. Arttrack: Articulated multi-person tracking in the wild. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. Cited on page 23.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. Cited on page 35.

J. Jaiganesh and M. Burtscher. A high-performance connected components implementation for GPUs. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pages 92–104, 2018. Cited on page 44.

A. Jaimovich, G. Elidan, H. Margalit, and N. Friedman. Towards an integrated protein–protein interaction network: A relational markov network approach. *Journal of Computational Biology*, 13(2):145–164, 2006. Cited on page 112.

J. Jain, J. Li, M. T. Chiu, A. Hassani, N. Orlov, and H. Shi. Oneformer: One transformer to rule universal image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2989–2998, 2023. Cited on page 39.

W. Jakob, J. Rhinelander, and D. Moldovan. pybind11 – Seamless operability between C++11 and Python, 2017. https://github.com/pybind/pybind11. Cited on page 112.

J. Jancsary and G. Matz. Convergent decomposition solvers for tree-reweighted free energies. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 388–398, 2011. Cited on page 74.

H. Jia, S. Ding, X. Xu, and R. Nie. The latest research progress on spectral clustering. *Neural Comput. Appl.*, 24(7–8):1477–1486, jun 2014. ISSN 0941-0643. doi: 10.1007/s00521-013-1439-2. Cited on page 56.

Y. Jin and J. F. JaJa. A high performance implementation of spectral clustering on CPU-GPU platforms. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, pages 825–834. IEEE, 2016. Cited on page 42.

J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019. Cited on pages 55 and 64.

J. K. Johnson, D. M. Malioutov, and A. S. Willsky. Lagrangian relaxation for MAP estimation in graphical models. *arXiv preprint arXiv:0710.0013*, 2007. Cited on page 74.

F. Jug, E. Levinkov, C. Blasse, E. W. Myers, and B. Andres. Moral lineage tracing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016. Cited on page 23.

J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. Cited on page 80.

D. Kainmueller, F. Jug, C. Rother, and G. Myers. Active graph matching for automatic joint segmentation and annotation of C. elegans. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 81–88. Springer, 2014. Cited on pages 92, 98, and 108.

J. Kappes, B. Andres, F. Hamprecht, C. Schnorr, S. Nowozin, D. Batra, S. Kim, B. Kausler, J. Lellmann, N. Komodakis, et al. A comparative study of modern inference techniques for discrete energy minimization problems. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1328–1335, 2013. Cited on pages 74 and 112.

J. H. Kappes, M. Speth, B. Andres, G. Reinelt, and C. Schn. Globally optimal image partitioning by multicuts. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer, 2011. Cited on pages 22 and 41.

J. H. Kappes, B. Savchynskyy, and C. Schnörr. A bundle approach to efficient map-inference by lagrangian relaxation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1688–1695. IEEE, 2012. Cited on page 74.

J. H. Kappes, B. Andres, F. A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, T. Kröger, J. Lellmann, N. Komodakis, B. Savchynskyy, and C. Rother. A comparative study of modern inference techniques for structured discrete energy minimization problems. *International Journal of Computer Vision*, 115(2):155–184, 2015. doi: 10.1007/s11263-015-0809-x. Cited on pages 74, 82, and 92.

J. H. Kappes, M. Speth, G. Reinelt, and C. Schnörr. Higher-order segmentation via multicuts. *Computer Vision and Image Understanding*, 143:104–119, 2016. Cited on page 23.

A. Kardoost and M. Keuper. Solving minimum cost lifted multicut problems by node agglomeration. In *Asian Conference on Computer Vision*, pages 74–89. Springer, 2018. Cited on pages 21, 50, 55, and 60.

A. Kazi, L. Cosmo, S.-A. Ahmadi, N. Navab, and M. M. Bronstein. Differentiable graph module (dgm) for graph convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1606–1617, 2022. Cited on page 55.

A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018. Cited on page 27.

B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2):291–307, 1970. Cited on page 22.

M. Keuper, E. Levinkov, N. Bonneel, G. Lavoué, T. Brox, and B. Andres. Efficient decomposition of image and mesh graphs by lifted multicuts. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015. Cited on pages 21, 22, 23, 30, 40, 41, 42, 44, 50, 51, 52, 55, 56, 57, 59, 63, and 114.

M. Keuper, S. Tang, B. Andres, T. Brox, and B. Schiele. Motion segmentation & multiple object tracking by correlation co-clustering. *IEEE transactions on pattern analysis and machine intelligence*, 42(1):140–153, 2018. Cited on page 23.

A. Khoreva, R. Benenson, J. Hosang, M. Hein, and B. Schiele. Simple does it: Weakly supervised instance and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 876–885, 2017. Cited on page 79.

S. Kim, S. Nowozin, P. Kohli, and C. D. Yoo. Higher-order correlation clustering for image segmentation. In *Advances in neural information processing systems*, 2011. Cited on pages 22 and 23.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Cited on pages 34 and 109.

A. Kirillov, D. Schlesinger, S. Zheng, B. Savchynskyy, P. H. S. Torr, and C. Rother. Joint training of generic cnn-crf models with stochastic optimization, 2016. Cited on page 27.

A. Kirillov, E. Levinkov, B. Andres, B. Savchynskyy, and C. Rother. Instancecut: from edges to instances with multicut. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. Cited on pages 23, 26, 30, and 65.

A. Kirillov, R. Girshick, K. He, and P. Dollár. Panoptic feature pyramid networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6399–6408, 2019a. Cited on pages 26 and 36.

A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár. Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9404–9413, 2019b. Cited on pages 24, 25, 26, 31, and 32.

A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár. Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9404–9413, 2019c. Cited on pages 65 and 66.

P. Knobelreiter, C. Sormann, A. Shekhovtsov, F. Fraundorfer, and T. Pock. Belief propagation reloaded: Learning bp-layers for labeling problems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7900–7909, 2020. Cited on page 27.

D. E. Knuth. *The art of computer programming, volume 4A: combinatorial algorithms, part 1*. Pearson Education India, 2011. Cited on page 78.

V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1568–1583, 2006. Cited on pages 72, 74, 76, and 92.

V. Kolmogorov. A new look at reweighted message passing. *IEEE transactions on pattern analysis and machine intelligence*, 37(5):919–930, 2014. Cited on pages 74 and 76.

V. Kolmogorov. Solving relaxations of map-mrf problems: Combinatorial in-face frank-wolfe directions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11980–11989, 2023. Cited on page 74.

N. Komodakis and G. Tziritas. Approximate labeling via graph cuts based on linear programming. *IEEE transactions on pattern analysis and machine intelligence*, 29(8):1436–1453, 2007. Cited on pages 74 and 93.

N. Komodakis, N. Paragios, and G. Tziritas. Mrf energy minimization and beyond via dual decomposition. *IEEE transactions on pattern analysis and machine intelligence*, 33(3):531–552, 2010. Cited on page 74.

J. Kotary, F. Fioretto, P. Van Hentenryck, and B. Wilder. End-to-end constrained optimization learning: A survey. *arXiv preprint arXiv:2103.16378*, 2021. Cited on page 27.

V. Kovalevsky and V. Koval. A diffusion algorithm for decreasing energy of max-sum labeling problem. *Glushkov Institute of Cybernetics, Kiev, USSR*, 1975. Cited on page 74.

T. Kroeger, J. H. Kappes, T. Beier, U. Koethe, and F. A. Hamprecht. Asymmetric cuts: Joint image labeling and partitioning. In *German Conference on Pattern Recognition*. Springer, 2014. Cited on pages 14, 23, 25, and 29.

J.-H. Lange. Multicut Optimization Guarantees & Geometry of Lifted Multicuts (Ph.D. thesis). 2020. Cited on page 22.

J.-H. Lange and P. Swoboda. Efficient message passing for 0–1 ILPs with binary decision diagrams. In *International Conference on Machine Learning*, pages 6000–6010. PMLR, 2021. Cited on pages 15, 70, 72, 75, 76, 77, 78, 81, 82, 83, 85, 91, 92, 94, and 104.

J.-H. Lange, A. Karrenbauer, and B. Andres. Partial optimality and fast lower bounds for weighted correlation clustering. In *International Conference on Machine Learning*, pages 2892–2901. PMLR, 2018. Cited on pages 20, 22, 41, 45, 50, 52, and 74.

J.-H. Lange, B. Andres, and P. Swoboda. Combinatorial persistency criteria for multicut and max-cut. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6093–6102, 2019. Cited on page 22.

E. Levinkov, A. Kirillov, and B. Andres. A comparative study of local search algorithms for correlation clustering. In *GCPR*, 2017a. Cited on pages 21, 42, 50, 51, and 55.

E. Levinkov, A. Kirillov, and B. Andres. A comparative study of local search algorithms for correlation clustering. In *German Conference on Pattern Recognition*. Springer, 2017b. Cited on page 40.

E. Levinkov, J. Uhrig, S. Tang, M. Omran, E. Insafutdinov, A. Kirillov, C. Rother, T. Brox, B. Schiele, and B. Andres. Joint graph decomposition & node labeling: Problem, algorithms, applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017c. Cited on page 23.

J. Li, A. Raventos, A. Bhargava, T. Tagawa, and A. Gaidon. Learning to fuse things and stuff. *arXiv preprint arXiv:1812.01192*, 2018a. Cited on page 26.

K. Li, K. Swersky, and R. Zemel. Efficient feature learning using perturb-and-map. In *Neural Information Processing Systems Workshop on Perturbations, Optimization, and Statistics*, 2013. Cited on page 27.

Q. Li, A. Arnab, and P. H. Torr. Weakly-and semi-supervised panoptic segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 102–118, 2018b. Cited on page 26.

Q. Li, X. Qi, and P. H. Torr. Unifying training and inference for panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13320–13328, 2020. Cited on pages 26, 27, and 36.

Y. Li, H. Takehara, T. Taketomi, B. Zheng, and M. Nießner. 4dcomplete: Non-rigid motion estimation beyond the observable surface. In *ICCV*, 2021. Cited on page 92.

T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2G014. Cited on page 34.

P. Lindenberger, P.-E. Sarlin, and M. Pollefeys. Lightglue: Local feature matching at light speed. *arXiv preprint arXiv:2306.13643*, 2023. Cited on page 79.

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989. Cited on pages 89 and 90.

Y. Liu, S. Yang, B. Li, W. Zhou, J. Xu, H. Li, and Y. Lu. Affinity derivation and graph merge for instance segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 686–703, 2018. Cited on page 26.

S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2): 129–137, 1982. doi: 10.1109/TIT.1982.1056489. Cited on pages 55 and 64.

E. M. Loiola, N. M. M. De Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European journal of operational research*, 176(2):657–690, 2007. Cited on pages 92 and 108.

F. Long, H. Peng, X. Liu, S. K. Kim, and E. Myers. A 3D digital atlas of C. elegans and its application to single-cell analyses. *Nature methods*, 6(9):667–672, 2009. Cited on page 108.

L. Lozano, D. Bergman, and J. C. Smith. On the consistent path problem. *Optimization Online e-prints*, 2018. Cited on page 83.

J. Lukasik, M. Keuper, M. Singh, and J. Yarkony. A benders decomposition approach to correlation clustering. In *2020 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC) and Workshop on Artificial Intelligence and Machine Learning for Scientific Applications (AI4S)*. IEEE, 2020. Cited on page 22.

R. Magnet, J. Ren, O. Sorkine-Hornung, and M. Ovsjanikov. Smooth non-rigid shape matching via effective dirichlet energy optimization. In *International Conference on 3D Vision (3DV)*, 2022. Cited on page 92.

C. Malin-Mayor, P. Hirsch, L. Guignard, K. McDole, Y. Wan, W. C. Lemon, D. Kainmueller, P. J. Keller, S. Preibisch, and J. Funke. Automated reconstruction of whole-embryo cell lineages by learning from sparse annotations. *Nature Biotechnology*, 41(1):44–49, 2023. Cited on page 79.

Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018. Cited on pages 60 and 64.

M. Maška, V. Ulman, P. Delgado-Rodriguez, E. Gómez-de Mariscal, T. Nečasová, F. A. Guerrero Peña, T. I. Ren, E. M. Meyerowitz, T. Scherr, K. Löffler, et al. The cell tracking challenge: 10 years of objective benchmarking. *Nature Methods*, pages 1–11, 2023. Cited on page 79.

D. A. McAllester, T. Hazan, and J. Keshet. Direct loss minimization for structured prediction. In *NIPS*, volume 1, page 3. Citeseer, 2010. Cited on pages 28 and 31.

T. Meltzer, A. Globerson, and Y. Weiss. Convergent message passing algorithms-a unifying view. *arXiv preprint arXiv:1205.2625*, 2012. Cited on page 74.

A. Mensch and M. Blondel. Differentiable dynamic programming for structured prediction and attention. In *International Conference on Machine Learning*, pages 3462–3471. PMLR, 2018. Cited on page 27.

L. Metz, J. Harrison, C. D. Freeman, A. Merchant, L. Beyer, J. Bradbury, N. Agrawal, B. Poole, I. Mordatch, A. Roberts, et al. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022. Cited on pages 99 and 115.

S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. Cited on page 26.

R. Mohan and A. Valada. Efficientps: Efficient panoptic segmentation. *International Journal of Computer Vision*, pages 1–29, 2021. Cited on page 26.

V. Monga, Y. Li, and Y. C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021. doi: 10.1109/MSP.2020.3016905. Cited on page 99.

MOSEK ApS. *9.0.105*, 2022. Cited on page 98.

V. Nair, S. Bartunov, F. Gimeno, I. von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020. Cited on pages 83, 98, and 99.

M. Naumov and T. Moon. Parallel spectral graph partitioning. *tech. rep., NVIDIA tech. rep*, 2016. Cited on page 42.

M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018. Cited on page 99.

J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980. Cited on page 89.

S. Nowozin and S. Jegelka. Solution stability in linear programming relaxations: Graph partitioning and unsupervised learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 769–776, 2009. Cited on page 41.

NVIDIA, P. Vingelmann, and F. H. Fitzek. CUDA, release: 11.2, 2021. URL https://developer.nvidia.com/cuda-toolkit. Cited on pages 49, 92, and 112.

S. Onn and L. J. Schulman. The vector partition problem for convex objective functions. *Mathematics of Operations Research*, 26(3):583–590, 2001. Cited on page 62.

M. Oosten, J. H. Rutten, and F. C. Spieksma. The clique partitioning problem: facets and patching facets. *Networks: An International Journal*, 38(4):209–226, 2001. Cited on page 20.

X. Pan, D. Papailiopoulos, S. Oymak, B. Recht, K. Ramchandran, and M. I. Jordan. Parallel correlation clustering on big graphs. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. Cited on page 55.

G. Papandreou and A. L. Yuille. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *2011 International Conference on Computer Vision*, pages 193–200. IEEE, 2011. Cited on page 27.

C. Pape. torch-em. https://github.com/constantinpape/torch-em, 2021. Cited on page 49.

C. Pape, T. Beier, P. Li, V. Jain, D. D. Bock, and A. Kreshuk. Solving large multicut problems for connectomics via domain decomposition. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017. Cited on pages 23, 40, 41, and 49.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. Cited on pages 34 and 111.

M. B. Paulus, D. Choi, D. Tarlow, A. Krause, and C. J. Maddison. Gradient estimation with stochastic softmax tricks. *arXiv preprint arXiv:2006.08063*, 2020. Cited on page 28.

M. B. Paulus, G. Zarpellon, A. Krause, L. Charlin, and C. Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International conference on machine learning*, pages 17584–17600. PMLR, 2022. Cited on page 98.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. Cited on page 64.

H. Peng, S. Thomson, and N. A. Smith. Backpropagating through structured argmax using a spigot. *arXiv preprint arXiv:1805.04658*, 2018. Cited on page 25.

K. Perumalla and M. Alam. Design Considerations for GPU-Based Mixed Integer Programming on Parallel Computing Platforms. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450384414. Cited on pages 81 and 82.

T. Plötz and S. Roth. Neural nearest neighbors networks. *Advances in Neural Information Processing Systems*, 31:1087–1098, 2018. Cited on page 27.

L. Porzi, S. R. Bulo, A. Colovic, and P. Kontschieder. Seamless scene segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8277–8286, 2019. Cited on pages 26 and 32.

A. Prouvost, J. Dumouchelle, L. Scavuzzo, M. Gasse, D. Chételat, and A. Lodi. Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020. Cited on pages 99 and 108.

R. Qaddoura, H. Faris, and I. Aljarah. An efficient clustering algorithm based on the k-nearest neighbors with an indexing ratio. *International Journal of Machine Learning and Cybernetics*, 11(3):675–714, 2020. Cited on page 55.

S. Qiao, L.-C. Chen, and A. Yuille. Detectors: Detecting objects with recursive feature pyramid and switchable atrous convolution. *arXiv preprint arXiv:2006.02334*, 2020. Cited on page 26.

R. Qiu, Z. Sun, and Y. Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. Cited on page 98.

T. Ralphs, Y. Shinano, T. Berthold, and T. Koch. Parallel solvers for mixed integer linear optimization. In *Handbook of parallel constraint reasoning*, pages 283–336. Springer, 2018. Cited on page 82.

D. Robert, H. Raguet, and L. Landrieu. Scalable 3d panoptic segmentation as superpoint graph clustering. *Proceedings of the IEEE International Conference on 3D Vision*, 2024. Cited on page 79.

B. Roessle and M. Nießner. End2end multi-view feature matching with differentiable pose optimization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 477–487, 2023. Cited on page 79.

P. Roetzer, P. Swoboda, D. Cremers, and F. Bernard. A scalable combinatorial solver for elastic geometrically consistent 3d shape matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 428–438, 2022. Cited on page 79.

P. Roetzer, A. Abbas, D. Cao, F. Bernard, and P. Swoboda. DiscoMatch: Fast Discrete Optimisation for Geometrically Consistent 3D Shape Matching. In *Proceedings of the European conference on computer vision (ECCV) (to appear)*, 2024. Cited on pages 15, 79, and 92.

M. Rolínek, V. Musil, A. Paulus, M. Vlastelica, C. Michaelis, and G. Martius. Optimizing rank-based metrics with blackbox differentiation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7620–7630, 2020a. Cited on page 28.

M. Rolínek, P. Swoboda, D. Zietlow, A. Paulus, V. Musil, and G. Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In *European Conference on Computer Vision*, pages 407–424. Springer, 2020b. Cited on pages 28 and 79.

B. Savchynskyy. Discrete graphical models—an optimization perspective. *Foundations and Trends® in Computer Graphics and Vision*, 11(3-4):160–429, 2019. Cited on pages 71, 74, and 75.

B. Savchynskyy, S. Schmidt, J. H. Kappes, and C. Schnörr. Efficient MRF energy minimization via adaptive diminishing smoothing. *UAI. Proceedings*, pages 746–755, 2012. 1. Cited on page 74.

L. Scavuzzo, F. Chen, D. Chételat, M. Gasse, A. Lodi, N. Yorke-Smith, and K. Aardal. Learning to branch with tree mdps. *Advances in Neural Information Processing Systems*, 35:18514–18526, 2022. Cited on page 99.

D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a SAT solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018. Cited on page 99.

A. Shekhovtsov, C. Reinbacher, G. Graber, and T. Pock. Solving dense image matching in real-time using discrete-continuous optimization. In *Proceedings of the 21st Computer Vision Winter Workshop (CVWW)*, page 13, 2016. ISBN 978-3-85125-388-7. Cited on pages 82 and 83.

Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1548–1554. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/214. Main Track. Cited on page 104.

E. Smith, J. Gondzio, and J. Hall. GPU acceleration of the matrix-free interior point method. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, editors, *Parallel Processing and Applied Mathematics*, pages 681–689, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-31464-3. Cited on page 83.

B. Sofranac, A. Gleixner, and S. Pokutta. Accelerating domain propagation: an efficient GPU-parallel algorithm over sparse matrices. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 1–11. IEEE, 2020. Cited on page 83.

J. Song, B. Andres, M. J. Black, O. Hilliges, and S. Tang. End-to-end learning for graph decomposition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019a. Cited on page 40.

J. Song, B. Andres, M. J. Black, O. Hilliges, and S. Tang. End-to-end learning for graph decomposition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10093–10102, 2019b. Cited on page 27.

N. Sonnerat, P. Wang, I. Ktena, S. Bartunov, and V. Nair. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*, 2021. Cited on pages 83 and 99.

K. Stinson, D. F. Gleich, and P. G. Constantine. A randomized algorithm for enumerating zonotope vertices. *arXiv preprint arXiv:1602.06620*, 2016. Cited on page 62.

G. Storvik and G. Dahl. Lagrangian-based methods for finding map solutions for mrf models. *IEEE Transactions on Image Processing*, 9(3):469–479, 2000. Cited on page 74.

Z. Sun and Y. Yang. DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization. *arXiv preprint arXiv:2302.08224*, 2023. Cited on page 98.

P. Swoboda and B. Andres. A message passing algorithm for the minimum cost multicut problem. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. Cited on pages 20, 22, 44, 45, and 74.

P. Swoboda and V. Kolmogorov. Map inference via block-coordinate frank-wolfe algorithm. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11146–11155, 2019. Cited on page 74.

P. Swoboda, J. Kuske, and B. Savchynskyy. A dual ascent framework for lagrangean decomposition of combinatorial problems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1596–1606, 2017a. Cited on pages 40, 41, 45, 46, 75, and 76.

P. Swoboda, C. Rother, H. Abu Alhaija, D. Kainmuller, and B. Savchynskyy. A study of lagrangean decompositions and dual ascent solvers for graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1607–1616, 2017b. Cited on pages 74 and 75.

P. Swoboda, A. Mokarian, C. Theobalt, F. Bernard, et al. A convex relaxation for multi-graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11156–11165, 2019. Cited on page 74.

P. Swoboda, A. Hornakova, P. Roetzer, and A. Abbas. Structured Prediction Problem Archive. *arXiv preprint arXiv:2202.03574*, 2022a. Cited on page 49.

P. Swoboda, A. Hornakova, P. Roetzer, B. Savchynskyy, and A. Abbas. Structured prediction problem archive. *arXiv preprint arXiv:2202.03574*, 2022b. Cited on pages 63, 92, 107, and 108.

S. Tang, M. Andriluka, B. Andres, and B. Schiele. Multiple people tracking by lifted multicut and person re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. Cited on page 23.

Z. Teed and J. Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 402–419. Springer, 2020. Cited on page 79.

Z. Tian, C. Shen, and H. Chen. Conditional convolutions for instance segmentation. *arXiv preprint arXiv:2003.05664*, 2020. Cited on pages 26 and 30.

C. Tjandraatmadja and W.-J. van Hoeve. Incorporating bounds from decision diagrams into integer programming. *Mathematical Programming Computation*, pages 1–32, 2020. Cited on page 84.

L. Torresani, V. Kolmogorov, and C. Rother. Feature correspondence via graph matching: Models and global optimization. In *European conference on computer vision*, pages 596–609. Springer, 2008. Cited on pages 74 and 92.

S. Tourani, A. Shekhovtsov, C. Rother, and B. Savchynskyy. MPLP++: Fast, parallel dual block-coordinate ascent for dense graphical models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. Cited on pages 46, 74, 82, 83, and 93.

S. Tourani, A. Shekhovtsov, C. Rother, and B. Savchynskyy. Taxonomy of dual block-coordinate ascent methods for discrete energy minimization. In *AISTATS*, 2020. Cited on pages 74 and 76.

M. Turner, T. Koch, F. Serrano, and M. Winkler. Adaptive cut selection in mixed-integer linear programming. *arXiv preprint arXiv:2202.10962*, 2022. Cited on pages 98 and 99.

J. Tönshoff, M. Ritzert, H. Wolf, and M. Grohe. Graph Neural Networks for Maximum Constraint Satisfaction. *Frontiers in Artificial Intelligence*, 3, 2021. ISSN 2624-8212. doi: 10.3389/frai.2020.580607. Cited on page 99.

V. Ulman, M. Maška, K. E. Magnusson, O. Ronneberger, C. Haubold, N. Harder, P. Matula, P. Matula, D. Svoboda, M. Radojevic, et al. An objective comparison of cell-tracking algorithms. *Nature methods*, 14(12):1141–1152, 2017. Cited on page 107.

N. Veldt. Correlation clustering via strong triadic closure labeling: Fast approximation algorithms and practical lower bounds. In *International Conference on Machine Learning*, pages 22060–22083. PMLR, 2022. Cited on page 55.

N. Veldt, A. I. Wirth, and D. F. Gleich. Correlation clustering with low-rank matrices. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1025–1034, 2017. Cited on pages 62 and 64.

V. Vineet and P. Narayanan. CUDA cuts: Fast graph cuts on the GPU. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008. Cited on page 83.

N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(95):2837–2854, 2010. Cited on page 64.

M. Vlastelica, A. Paulus, V. Musil, G. Martius, and M. Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019. Cited on pages 25, 28, 30, 31, 35, 36, and 38.

U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007. Cited on page 56.

H. Wang and D. Koller. Subproblem-tree calibration: A unified approach to max-product message passing. In *ICML (2)*, pages 190–198, 2013. Cited on page 74.

H. Wang, Y. Zhu, H. Adam, A. Yuille, and L.-C. Chen. Max-deeplab: End-to-end panoptic segmentation with mask transformers. *arXiv preprint arXiv:2012.00759*, 2020a. Cited on pages 25, 26, 27, 30, 36, 39, 79, and 80.

H. Wang, Y. Zhu, B. Green, H. Adam, A. Yuille, and L.-C. Chen. Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. In *European Conference on Computer Vision*, pages 108–126. Springer, 2020b. Cited on pages 26, 36, and 39.

H. Wang, Y. Zhu, H. Adam, A. Yuille, and L.-C. Chen. Max-deeplab: End-to-end panoptic segmentation with mask transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5463–5474, 2021a. Cited on page 65.

P.-W. Wang, P. Donti, B. Wilder, and Z. Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pages 6545–6554. PMLR, 2019. Cited on page 27.

R. Wang, J. Yan, and X. Yang. Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5261–5279, 2021b. Cited on page 99.

W. Wang, M. Feiszli, H. Wang, J. Malik, and D. Tran. Open-world instance segmentation: Exploiting pseudo ground truth from learned pairwise affinity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4422–4432, 2022. Cited on page 79.

X. Wang, R. Girdhar, S. X. Yu, and I. Misra. Cut and learn for unsupervised object detection and instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3124–3134, 2023. Cited on page 79.

M. Weber, H. Wang, S. Qiao, J. Xie, M. D. Collins, Y. Zhu, L. Yuan, D. Kim, Q. Yu, D. Cremers, L. Leal-Taixe, A. L. Yuille, F. Schroff, H. Adam, and L.-C. Chen. DeepLab2: A TensorFlow Library for Deep Labeling. *arXiv: 2106.09748*, 2021. Cited on page 65.

D. Wedelin. An algorithm for large scale 0–1 integer programming with application to airline crew scheduling. *Annals of operations research*, 57(1):283–301, 1995a. Cited on page 90.

D. Wedelin. The design of a 0–1 integer optimizer and its application in the carmen system. *European journal of operational research*, 87(3):722–730, 1995b. Cited on page 90.

W. H. Wen-mei. *GPU Computing Gems Jade Edition*. Elsevier, 2011. Cited on page 44.

T. Werner. A linear programming approach to max-sum problem: A review. *IEEE transactions on pattern analysis and machine intelligence*, 29(7):1165–1179, 2007. Cited on pages 71, 74, 88, and 104.

T. Werner, D. Průša, and T. Dlask. Relative interior rule in block-coordinate descent. In *Proceedings of the IEEE International Conference on Computer Vision*, 2020. Cited on pages 76, 85, 88, 95, and 104.

B. Wilder, E. Ewing, B. Dilkina, and M. Tambe. End to end learning and optimization on graphs. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. Cited on page 27.

T. Windheuser, U. Schlickewei, F. R. Schmidt, and D. Cremers. Geometrically consistent elastic matching of 3d shapes: A linear programming solution. In *2011 International Conference on Computer Vision*, pages 2134–2141. IEEE, 2011a. Cited on page 112.

T. Windheuser, U. Schlickwei, F. R. Schimdt, and D. Cremers. Large-scale integer linear programming for orientation preserving 3d shape matching. In *Computer Graphics Forum*, volume 30, pages 1471–1480. Wiley Online Library, 2011b. Cited on pages 79 and 112.

S. Wolf, Y. Li, C. Pape, A. Bailoni, A. Kreshuk, and F. A. Hamprecht. The semantic mutex watershed for efficient bottom-up semantic instance segmentation. In *European Conference on Computer Vision*, pages 208–224. Springer, 2020. Cited on pages 25, 26, 35, 36, 38, and 41.

J. Wu, Z. He, and B. Hong. Chapter 5 - efficient CUDA algorithms for the maximum network flow problem. In W. mei W. Hwu, editor, *GPU Computing Gems Jade Edition*, Applications of GPU Computing Series, pages 55–66. Morgan Kaufmann, Boston, 2012. ISBN 978-0-12-385963-1. Cited on page 83.

Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. Cited on page 34.

Y. Wu, W. Song, Z. Cao, and J. Zhang. Learning large neighborhood search policy for integer programming. *Advances in Neural Information Processing Systems*, 34, 2021. Cited on page 99.

Z. Wu, Y. Xiong, S. X. Yu, and D. Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018. Cited on page 26.

Y. Xie, H. Dai, M. Chen, B. Dai, T. Zhao, H. Zha, W. Wei, and T. Pfister. Differentiable top-k operator with optimal transport. *arXiv preprint arXiv:2002.06504*, 2020. Cited on page 27.

L. Xin, W. Song, Z. Cao, and J. Zhang. NeuroLKH: Combining Deep Learning Model with Lin-Kernighan-Helsgaun Heuristic for Solving the Traveling Salesman Problem. *Advances in Neural Information Processing Systems*, 34, 2021. Cited on page 99.

Y. Xiong, R. Liao, H. Zhao, R. Hu, M. Bai, E. Yumer, and R. Urtasun. Upsnet: A unified panoptic segmentation network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8818–8826, 2019. Cited on pages 26, 30, 32, and 36.

Z. Xu, T. Ajanthan, and R. Hartley. Fast and differentiable message passing on pairwise markov random fields. In *Proceedings of the Asian Conference on Computer Vision*, 2020. Cited on pages 82 and 83.

T.-J. Yang, M. D. Collins, Y. Zhu, J.-J. Hwang, T. Liu, X. Zhang, V. Sze, G. Papandreou, and L.-C. Chen. Deeperlab: Single-shot image parser. *arXiv preprint arXiv:1902.05093*, 2019. Cited on pages 26, 28, and 35.

Y. Yang, H. Li, X. Li, Q. Zhao, J. Wu, and Z. Lin. Sognet: Scene overlap graph network for panoptic segmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12637–12644, 2020a. Cited on page 26.

Y. Yang, J. Sun, H. Li, and Z. Xu. ADMM-CSNet: A Deep Learning Approach for Image Compressive Sensing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42 (3):521–538, 2020b. doi: 10.1109/TPAMI.2018.2883941. Cited on page 99.

Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan. Mvsnet: Depth inference for unstructured multi-view stereo. In *Proceedings of the European conference on computer vision (ECCV)*, pages 767–783, 2018. Cited on page 79.

J. Yarkony, A. Ihler, and C. C. Fowlkes. Fast planar correlation clustering for image segmentation. In *European Conference on Computer Vision*. Springer, 2012. Cited on pages 22 and 23.

J. Yarkony, T. Beier, P. Baldi, and F. A. Hamprecht. Parallel multicut segmentation via dual decomposition. In *International Workshop on New Frontiers in Mining Complex Patterns*, pages 56–68. Springer, 2014. Cited on page 20.

Q. Yu, H. Wang, S. Qiao, M. Collins, Y. Zhu, H. Adam, A. Yuille, and L.-C. Chen. k-means mask transformer. In *European Conference on Computer Vision*, pages 288–307. Springer, 2022. Cited on page 65.

X. Zeng, R. Liao, L. Gu, Y. Xiong, S. Fidler, and R. Urtasun. Dmm-net: Differentiable mask-matching network for video object segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3929–3938, 2019. Cited on page 27.

Y. Zhang, D. W. Zhang, S. Lacoste-Julien, G. J. Burghouts, and C. G. Snoek. Unlocking slot attention by changing optimal transport costs. *arXiv preprint arXiv:2301.13197*, 2023. Cited on page 79.

Z. Zhang, Q. Shi, J. McAuley, W. Wei, Y. Zhang, and A. Van Den Hengel. Pairwise matching through max-weight bipartite belief propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1202–1210, 2016. Cited on page 74.

S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1529–1537, 2015. Cited on page 27.

Z. Zheng, J. S. Lauritzen, E. Perlman, C. G. Robinson, M. Nichols, D. Milkie, O. Torrens, J. Price, C. B. Fisher, N. Sharifi, et al. A complete electron microscopy volume of the brain of adult Drosophila melanogaster. *Cell*, 174(3):730–743, 2018. Cited on page 49.