Lifted Edges as Connectivity Priors for Multicut and Disjoint Paths

A dissertation submitted towards the degree
Doctor of Natural Sciences
(Dr. rer. nat.)
of the Faculty of Mathematics and Computer Science
of Saarland University

by
Ing. Andrea Horňáková

Saarbrücken 2022

Day of Colloquium	29th of June, 2022

Dean of the Faculty	Prof. Dr. Jürgen Steimle Saarland University, Germany

Examination Committee

Chair	Prof. Dr. Jan Reineke,
Reviewer, Advisor	Dr. Paul Swoboda
Reviewer 1	Prof. Dr. Bernt Schiele
Reviewer 2	doc. Ing. Tomáš Werner, Ph.D.
Academic assistant	Dr. Jan Eric Lenssen

ABSTRACT

This work studies graph decompositions and their representation by 0/1 labeling of edges. We study two problems. The first is multicut (MC) which represents decompositions of undirected graphs (clustering of nodes into connected components). The second is disjoint paths (DP) in directed acyclic graphs where the clusters correspond to node-disjoint paths. Unlike an alternative representation by node labeling, the number of clusters is not part of the input but is fully determined by the costs of edges.

Our main interest is to study connectivity priors represented by so-called lifted edges in the two problems. The cost of a lifted edge expresses whether its endpoints should belong to the same cluster (path) in the optimal decomposition. We call the resulting problems lifted multicut (LMC) and lifted disjoint paths (LDP). The extension of MC to LMC was originally motivated by image segmentation where the information about the connectivity between non-neighboring pixels or superpixels led to a significant quality improvement. After that, LMC was successfully applied to other problems like multiple object tracking (MOT) which is also the main application of our proposed LDP model.

Our study of lifted multicut concentrates on partial LMC represented by labeling of a subset of (lifted) edges. Given partial labeling, we conclude that deciding whether a complete LMC consistent with the partial labels exists is NP-complete. Similarly, we conclude that deciding whether an unlabeled edge exists such that its label is determined by the labels of other edges is NP-hard. After that, we present metrics for comparing (partial) graph decompositions. Finally, we study the properties of the LMC polytope.

The largest part of this work is dedicated to the proposed LDP problem. We prove that this problem is NP-hard and propose an optimal integer linear programming (ILP) solver. In order to enable its global optimization, we formulate several classes of linear inequalities that produce a high-quality LP relaxation. Additionally, we propose efficient cutting plane algorithms for separating the proposed linear inequalities.

Despite the advanced constraints and efficient separation routines, the general time complexity of our optimal ILP solver remains exponential. In order to solve even larger instances, we introduce an approximate LDP solver based on Lagrange decomposition.

LDP is a convenient model for MOT because the underlying disjoint paths model naturally leads to trajectories of objects. Moreover, lifted edges encode long-range temporal interactions and thus help to prevent id switches and re-identify persons. Our tracker using the optimal LDP solver achieves nearly optimal assignments w.r.t. input detections. Consequently, it was a leading tracker on three benchmarks of the MOT challenge MOT15/16/17, improving significantly over state-of-the-art at the time of its publication. Our approximate LDP solver enables us to process the MOT15/16/17 benchmarks without sacrificing solution quality and allows for solving large and dense instances of a challenging dataset MOT20. On all these four standard MOT benchmarks we achieved performance comparable or better than state-of-the-art methods (at the time of publication) including our tracker based on the optimal LDP solver.

ZUSAMMENFASSUNG

Diese Arbeit studiert Graphenzerlegungen und ihre Repräsentation durch 0/1-wertige Kantenbelegungen. Das erste Problem ist das Mehrfachschnittproblem. Es repräsentiert Zerlegungen von ungerichteten Graphen (Cluster von Knoten sodass jeder Cluster eine Zusammenhangskomponente repräsentiert). Das zweite Problem ist die Suche von disjunkten Pfaden in einem gerichteten azyklischen Graph in dem die Cluster knotendisjunkten Pfaden entsprechen. Im Unterschied zu der alternativen Repräsentation durch Knotenbelegungen ist die Zahl von Clustern nicht im Voraus gegeben, sondern sie ist abhängig von den Kosten der Kanten.

Der Fokus dieser Arbeit ist die Erforschung von hochgezogenen Kannten, die eine a-priori Information über Verbundenheit von Knoten in Clustern respektive durch Pfade in den zwei Problemen darstellen. Die Kosten einer hochgezogenen Kante drücken aus, ob ihre Knoten zu dem gleichen Cluster (Pfad) in der optimalen Zerlegung gehören sollten. Wir bezeichnen diese neuen Probleme als das hochgezogene Mehrfachschnittproblem und das Problem der hochgezogenen disjunkten Pfade. Die Erweiterung des Mehrfachschnittproblems zu dem hochgezogenen Mehrfachschnittproblem wurde ursprünglich durch die Bildsegmentierung motiviert, für die die Information über Verbundenheit von nicht benachbarten Pixeln oder Superpixeln zu einer bedeutenden Verbesserung der Qualität führte. Danach wurde das hochgezogene Mehrfachschnittproblem zu der Lösung von anderen Problemen wie zum Beispiel der Verfolgung von mehreren Objekten in einem Video angewendet. Diese Aufgabe ist auch die Hauptanwendung des vorgeschlagenen Problems der hochgezogenen disjunkte Pfade.

In unserer Untersuchung des hochgezogenen Mehrfachschnittproblems konzentrieren wir uns auf das teilweise hochgezogene Mehrfachschnittproblem. Das Problem wird durch eine Belegung einer Teilmenge der (hochgezogenen) Kanten repräsentiert. Wir beweisen, dass es NP-vollständig ist zu entscheiden, ob ein kompletter hochgezogener Mehrfachschnitt existiert, der einer gegebenen teilweisen Kantenbezeichnung entspricht. In analoger Weise beweisen wir, dass es NP-schwer ist zu entscheiden, ob eine nicht belegte Kante existiert, deren Belegung durch die Belegungen anderer Kanten entschieden ist. Danach präsentieren wir Metriken zum Vergleich von (teilweisen) Graphenzerlegungen. Schließlich untersuchen wir Eigenschaften des hochgezogenen Mehrfachschnitt-Polytops.

Der größte Teil dieser Arbeit widmet sich dem von uns vorgeschlagenen Problem der hochgezogenen disjunkten Pfade. Wir beweisen, dass es NP-schwer ist. Wir formulieren es als ein ganzzahliges lineares Optimierungsproblem und implementieren ein Programm für dessen optimale Lösung. Um die globale Optimierung zu ermöglichen, formulieren wir mehrere Klassen von linearen Ungleichungen, die zu einer linearen Relaxierung mit einer hohen Qualität führen. Zusätzlich präsentieren wir ein effektives Schnittebenenverfahren für die Separierung der vorgeschlagenen Ungleichungen.

Trotz der fortgeschrittenen Ungleichungen und der Effizienz der Schnittebenenseparierung in unserem optimalen Löser bleibt die allgemeine Komplexität des Algorithmus exponentiell. Um noch kompliziertere Instanzen zu lösen, präsentieren wir einen approximativen Löser, der auf Lagrange-Dualität aufbaut.

Hochgezogene disjunkte Pfade sind ein praktisches Modell für die Verfolgung von mehreren Objekten, weil die disjunkten Pfade eine natürliche Repräsentation von Trajektorien der Objekten darstellen. Außerdem repräsentieren die hochgezogenen Kanten Interaktionen einer langen zeitlichen Reichweite. Deswegen helfen sie dieselbe Person in zeitlich weiter auseinander liegenden Zeitpunkten wieder zu identifizieren und Verwechselungen ihrer Identität zu verhindern. Aus diesem Grund war unsere Methode zur Zeit ihrer Publikation die beste für drei Vergleichsdatensätzen MOT Challenge MOT15/16/17 für die Verfolgung von mehreren Objekten. Im Vergleich zu den bisherigen besten Methoden war ihre Leistung sogar bedeutend höher. Unsere approximative Methode für hochgezogene disjunkte Pfade ermöglicht uns die Vergleichsdatensätzen MOT15/16/17 zu verarbeiten ohne die Qualität der Lösungen zu vermindern und erlaubt uns, die großen Instanzen mit hoher Personendichte des anspruchsvolleren Datensatzes MOT20 zu lösen. Zur Zeit ihrer Publikation erreichte die Methode vergleichbare oder bessere Ergebnisse als die bisherigen besten Methoden einschließlich unseres optimalen Löser für hochgezogene disjunkte Pfade.

ACKNOWLEDGEMENTS

Hereby, I would like to thank all people who supported me and helped me during my PhD study. In the first place, I would like to thank my supervisor Paul Swoboda for his patient and productive supervision, good guidance, and support. He always took time for a discussion. He supported me when I needed special working conditions due to my family duties, especially during the covid pandemic.

I would like to thank all my co-authors for a pleasant and constructive cooperation. Besides my supervisor Paul Swoboda, I would like to name especially Roberto Henschel, Timo Kaiser, Bjoern Andres, and Jan-Hendrik Lange for their major contribution to the shared publications that are part of this thesis. The publications could not be realized without their part of the work. I would like to thank Bjoern Andres for his supervision and help during the work on our common paper. I would like to mention also Michal Rolinek who helped us with our latest publication. I would like to thank Jiles Vreeken, Marcel Schulz and Markus List who cooperated with me on a research project that is not part of this thesis.

I am very grateful to Bernt Schiele, the director of our department, who provided me with good working conditions, fully supported me in combining my working duties with family, and found a solution in the difficult stage of my PhD study by finding a new supervisor. Also, other people at MPI and Saarland University helped me to organize my work and family life and helped me with administrative issues. These were: Connie Balzert, Alexandra Klasen-Schmitt and Michael Bentz. Michelle Carnell and Susanne Vohl from the Graduate School help me, especially at the beginning of my PhD study. I would like to thank my colleagues for creating a friendly working environment.

This thesis could never be finished without many members of my family who supported me during my whole study and helped me with my family duties in the busy working times. For these reasons, a special thank belongs to my husband, my mother, my father, my mother in law and the grandmother of my husband. I would like to thank my daughter for her patience and for bringing joy in my life. I am also grateful to my friends, my sister, and other family members who provided me mental support.

CONTENTS

1 INTRODUCTION

GRAPHS are useful and powerful structures that enable us to model pairwise relationships between objects. Among others, they play a crucial role in many tasks in machine learning and computer vision. A special interest of this work is studying decompositions of graph nodes into connected components (graph clustering) and the applications of the presented methods in computer vision problems.

A useful tool for studying graph decompositions is multicut. It is a representation of graph decompositions by sets of edges that straddle distinct components. If each edge is assigned a real cost, the optimal graph decomposition is the one that minimizes the sum of costs of multicut edges. One advantage of the multicut problem over other clustering formulations is that the number of clusters is determined purely by the edge costs instead of being known in advance.

These properties make multicut a convenient model for example for image segmentation. Here, graph nodes represent pixels resp. superpixels of an image and edges are placed between those pixels or superpixels that are adjacent. Consequently, the decomposition of such a graph is a natural representation for image segmentation. In this case, each cluster corresponds to one object in the image.

The longest part of this work is dedicated to studying a special class of decompositions where the graphs are directed and the node clusters are constrained to be node-disjoint paths in the given graph. This special case of the network flow problem is a natural model for multiple object tracking (MOT) in a video. Here, objects in each video frame are represented by graph vertices and edges connect objects between video frames. The aim is to find a set of node-disjoint paths such that one path corresponds to the trajectory of one object.

In the above-stated examples of graph partitioning algorithms in computer vision, it is often helpful to employ connectivity preferences of object pairs that are not connected by an edge. For instance, in the task of image segmentation, there is an ambiguity about the classification of pixels close to the boundary between objects. On the other hand, we can classify accurately the majority of the pixels that are far from the object boundaries. Consequently, connectivity preferences cannot be always fully captured by the costs of edges between neighboring pixels (or superpixels) and information about the connectivity of non-neighboring pixels can improve the segmentation accuracy.

Alternatively, consider two non-consecutive video frames in the task of multiple object tracking. We can typically identify pairs of object detections that represent either the same object captured in different time frames of the video or two different objects. In the DP formulation, there are so-called skip edges between objects in non-consecutive time frames. However, their cost contributes to the objective value only if the respective object detections directly follow each other in a trajectory. Therefore, it is beneficial to extend the model via pairwise connectivity priors that contribute to the objective value whenever the respective object detections are connected via an arbitrary trajectory. Including this higher-order information into our decision procedure leads to more stable trajectories and thus it increases the precision of DP significantly, especially in cases where objects were occluded or not detected for some time.

Based on this motivation, we concentrate on two special aspects of graph decompositions. We study incorporating connectivity priors between non-neighboring graph nodes on the one hand and working with partial information on the other hand. First, we study the enhancements of the basic multicut and the disjoint paths problem with so-called lifted edges that represent connectivity priors of non-neighboring node pairs. In particular, the cost of a lifted edge indicates whether two objects represented by the edge's endpoints should belong to the same cluster (path) or not. The two models are called lifted multicut and lifted disjoint paths. Second, we inspect the possibility to work only with partial information about the graph decomposition represented by lifted multicut. That is, we are interested in the scenarios where only some of the (lifted) graph edges are marked as must-join or must-cut (multicut) edges while the rest of the edges is unlabeled.

1.1 OUTLINE AND CONTRIBUTIONS

ANALYSIS OF GRAPH DECOMPOSITIONS BY LIFTED MULTICUTS

We study the set of all decompositions (clusterings) of a graph through its characterization as a set of lifted multicuts. This leads us to practically relevant insights related to the definition of classes of decompositions by must-join and must-cut constraints and related to the comparison of clusterings by metrics. We also establish properties of some facets of lifted multicut polytope.

This chapter contains a substantial part of our work published in Horňáková *et al.* (2017) which was a joint work of the author of this thesis, Jan-Hendrik Lange and Bjoern Andres. In particular, Bjoern Andres supervised the work and suggested the concepts that led to the author's results presented in Sections 3.4 and 3.5. He also contributed the whole content of Section 3.3 (except the previously known facts) and Section 3.6. His theorem about facet defining cut inequalities was extended by Jan-Hendrik Lange and the thesis author (a generalization of Point 2).

Section 3.3. This section contains the basic definitions of the multicut problem as introduced and studied by Chopra and Rao (1993) and Deza *et al.* (1991). After that, we present its extension to the lifted multicut problem.

Section 3.4. In this section, we generalize the lifted multicut to the partial lifted multicut where each graph edge can be labeled either as must-join or must-cut or unlabeled. Using the concept of unlabeled edges, two natural questions arise. The first is consistency. That is, is there labeling of the unlabeled edges such that the result is a valid lifted multicut? The second is specificity. That is, is there an unlabeled edge that must be labeled as join or cut based on the labels of other edges? We conclude that the problems are NP-complete resp. NP-hard in general. We also provide special cases when the problems can be decided efficiently.

Section 3.5. Inspired by the concept of partial edge labeling, we study the possible metrics for the comparison of two (partial) graph clusterings. They can be viewed as a generalization of two commonly used graph metrics. These are the Hamming metric (Hamming, 1950) and Rand's metric (Rand, 1971). The main idea is that we allow comparing labels of a subset of the (lifted) graph edges instead of the labels of all edges.

Section 3.6. In this section, we study the lifted multicut polytope. We first show that its dimension is equal to the number of edges in the lifted graph. Second, we provide several necessary conditions for lifted multicut cut inequalities to be facet defining. The original publication Horňáková *et al.* (2017) contains more results on this topic.

LIFTED DISJOINT PATHS WITH APPLICATION IN MULTIPLE OBJECT TRACKING

Disjoint paths problem (DP) is a natural model for multiple object tracking (MOT) where trajectories of objects are represented by node-disjoint paths. This model is a special case of the network flow problem where the flow through each edge and through each node is restricted to be zero or one. The problem can be solved efficiently e.g. by the method from Kovács (2015) but it is prone to identity switches because it evaluates only connections between detections that directly follow each other in a trajectory.

In this chapter, we enhance the disjoint paths problem by lifted edges that represent connectivity priors. Consequently, the similarities of arbitrary pairs of objects that belong to the same path can influence the objective value but do not influence the set of feasible solutions given by the underlying DP model. Therefore, lifted edges help to preserve the identities of objects, prevent ID-switches and thus ensure more stable trajectories. We call the resulting model lifted disjoint paths (LDP).

Even though (LDP) is NP-hard (which we prove), we develop an optimal ILP solver for it. We would like to mention that the previous state-of-the-art trackers based on combinatorial models either employ heuristic solvers or are limited in the integration of long-range information, in contrast to our work. Our tracking framework based on the optimal LDP solver increased the stability of object trajectories and led to state-of-the-art results on three standard MOT benchmarks MOT15/16/17 (Leal-Taixé *et al.*, 2015; Milan *et al.*, 2016) at the time of its publication, significantly outperforming previous methods.

This chapter contains our work published in Horňáková *et al.* (2020) which was a joint work of the author of this thesis, Roberto Henschel and Paul Swoboda with the help of Bodo Rosenhahn. Roberto Henschel carried out all the experimental work, cost learning, preprocessing, and post-processing. In general, he implemented all parts of the tracking framework except the LDP solver which is the work of the author. He also proposed the two-step procedure which was implemented by the author.

Section 4.3. This section presents the formulation of the (LDP) model as an enhancement of (DP) by lifted edges.

Section 4.4 In this section, we present an ILP formulation for solving (LDP). We first provide the classical flow conservation constraints (Dantzig and Fulkerson, 1955). Then, we propose a class of advanced ILP constraints ensuring that a lifted edge is active if and only if there is a flow between its endpoints in the flow network. We analyze the tightness of the constraints and prove among others that the proposed path inequalities are tighter than analogical inequalities for lifted multicut. Their high quality plays an important role in our exact ILP solver.

Section 4.5. Our solver uses the state-of-the-art ILP solver Gurobi (Gurobi Optimization, 2019). Since there are exponentially many ILP constraints proposed in Section 4.4, we add them in a cutting plane manner during the branch and cut phase of the ILP solver. That is, whenever a candidate feasible solution is found, we add to the set of problem inequalities those that are violated by the solution. Alternatively, we store the solution as a new upper bound if no inequalities are violated. This section provides efficient algorithms for separating the violated inequalities.

Section 4.6. In this section, we prove that the lifted disjoint paths problem is NP-hard.

Section 4.7. This section describes the experiments, graph construction, and a two-step procedure needed for solving large data. It also details the whole tracking pipeline including preprocessing, post-processing, edge cost learning, and experimental setup in general. We demonstrate that our method outperforms other trackers on three standard MOT benchmarks and provide some ablation studies.

AN EFFICIENT APPROXIMATE SOLVER FOR LIFTED DISJOINT PATHS

The solver presented in Chapter 4 achieved state-of-the-art results on three MOT benchmark datasets. However, despite high-quality ILP constraints and efficient separation routines for them that make the solver able to solve real-world problems efficiently, its general runtime remains exponential. Therefore, we decided to develop an approximate LDP solver with polynomial runtime that would enable LDP-based tracking for long and crowded sequences. The resulting tracker achieved on four standard MOT benchmarks comparable or better results with the state-of-the-art methods (at the time of its publication) including our tracker from Chapter 4. Our LDP solver is based on Lagrangean (dual) decomposition and thus delivers not only an approximate solution but also its gap to the optimum. Therefore, it is a principled approach that is independent of any commercial solver like Gurobi (Gurobi Optimization, 2019).

This chapter contains our work published in Hornakova *et al.* (2021) which is a joint work of Andrea Hornakova, Timo Kaiser, Paul Swoboda, Michal Rolinek, and Roberto Henschel with the help of Bodo Rosenhahn. The implementation of all the tracking components (edge cost learning, preprocessing, post-processing etc.) except the LDP solver is the work of Timo Kaiser with the help of Roberto Henschel and Michal Rolinek. The experiments and evaluations were done by Timo Kaiser and the author. The author implemented the LDP solver within the message passing framework provided by Swoboda *et al.* (2017a).

Sections 5.3 and 5.4. The base for our approximate solver is a message passing algorithm using the Lagrange decomposition framework developed by Swoboda *et al.* (2017a). This decomposition of the LDP to small subproblems provides a dual task and hence a lower bound to the original problem. The lower bound is constantly improved by the message passing. In combination with a suitable primal heuristic using the dual costs, the approximate solver is able to provide a high-quality approximate solution together with the gap to the optimum. These sections provide details about the decomposition itself specific to LDP. We describe the classes of LDP subproblems including their minimization procedures and algorithms for message passing. We also describe our primal heuristic.

Section 5.5. Here, we analyze the theoretical complexity of the used algorithms and conclude that the proposed LDP solver has polynomial time complexity.

Section 5.6. We incorporated our approximate LDP solver into a new tracking framework. This section describes all the details about the complete tracking system and provides experimental evaluation. This tracker has several differences in comparison to the one presented in Chapter 4 enabling its scalability to large data with crowded scenarios of the challenging MOT20 dataset (Dendorfer *et al.*, 2020). For completeness, we also perform experiments on the other three standard MOT benchmarks MOT15/16/17 (Leal-Taixé *et al.*, 2015; Milan *et al.*, 2016). We demonstrate that the tracker achieves comparable or better results in comparison to the state-of-the-art methods. Finally, we provide a runtime comparison with the solver from Chapter 4.

BACKGROUND AND RELATED WORK 2

2.1 MULTICUT

We start with basic definitions of the *multicut* problem and its relation to graph decomposition. More details can be found in Section 3.3. The decomposition of a graph $G = (V, E)$ is a partition Π of the node set V such that, for every subset $U \in \Pi$ of nodes, the subgraph of G induced by U is connected. Each partition can be represented by multicut which is a set of edges that straddle distinct components. For any graph G, a one-to-one relation exists between the decompositions and the multicuts of G. An example of a graph decomposition together with the multicut representing it is in Figure 2.1.

Multicuts of graph G are defined by means of cycles in G. If $M \subseteq E$ is a multicut of G, there cannot be a cycle in G that contains exactly one edge from M. We can represent each set $M \subseteq E$ by its characteristic function $x \in \{0,1\}^E$ where $M = \{e \in E | x_e = 1\}$. Then M is a multicut of G iff for its characteristic function $x \in \{0,1\}^E$ holds

$$\forall C \in \text{cycles}(G)\ \forall e \in C:\ x_e \leq \sum_{e' \in C \setminus \{e\}} x_{e'} . \tag{2.1}$$

We use notation $X_G = \{x \in \{0,1\}^E | x \text{ satisfies } (2.1)\}$. That is, X_G is the set of characteristic functions of multicuts of G.

Definition 2.1. (MC) *For any connected graph $G = (V, E)$, and any $c : E \to \mathbb{R}$, the instance of the* minimum cost multicut problem *w.r.t. G and c is the optimization problem*

$$\min_{x \in X_G} \sum_{e \in E} c_e x_e . \tag{MC}$$

Polyhedral study. The two initial publications Grötschel and Wakabayashi (1989) and Grötschel and Wakabayashi (1990) studied multicut polytope of a complete graph. They referred to this polytope as clique partitioning polytope and mentioned that the clique partitioning of a complete graph can be viewed as multicut.

The work from Chopra and Rao (1993) explored the multicut polytope of general graphs. They referred to the studied problem as the partition problem. Among the most important contributions of this work is proposing the cycle inequalities (2.1), wheel inequalities, and bicycle wheel inequalities for the multicut problem and providing criteria under which those inequalities are facet defining. These findings are very important for instance for cutting-plane algorithms solving the problem (MC). Another important

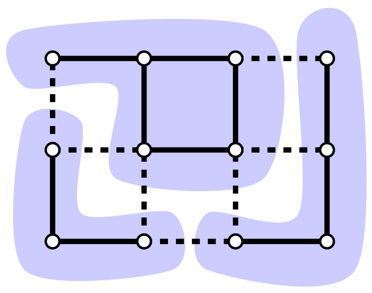


Figure 2.1: A decomposition of a graph is a partition of the node set into connected components. Any decomposition is characterized by the set of edges that straddle distinct components (dashed lines). This set of edges is called multicut.

contribution is proving that the dimension of multicut polytope equals to the number of graph edges.

The authors Deza *et al.* (1992) extended the class of inequalities valid for the multicut polytope by a large class of clique-web inequalities and provided a complex characterization of those that define facets. They referred to the studied problem as multicut or multicut with k shores. They also studied the multicut polytope together with equicut, balance cut, etc. in Deza *et al.* (1991). Additional results including the findings of the two publications can be found in the book from Deza and Laurent (1997).

2.1.1 PROBLEMS EQUIVALENT TO MULTICUT

Classical multicut. A definition of multicut different from Definition 2.1 was adopted by Garg *et al.* (1997), Vazirani (2001) and Schrijver (2003). The problem was introduced by Hu (1969) as a dual problem to maximum multi-commodity flow. In compliance with Lange (2020), we refer to this alternative definition as *classical multicut*.

Although the classical multicut Definition 2.2 is different from Definition 2.1 that we adopt in this thesis, the two problems are actually equivalent as we discuss later in this section.

Definition 2.2. *(Classical multicut) Given a graph $G = (V, E)$ with a positive weight on every edge $c : E \to \mathbb{R}^+$, and a list of node pairs, $(s_1, t_1), \dots, (s_k, t_k)$, find a minimum weight set of edges separating each pair of nodes vertices in the list. We call such a set of edges a multicut.*

Correlation clustering. The multicut problem is related to correlation clustering introduced by Bansal *et al.* (2004). Here, all edges of a complete graph are labeled either with + or - and the goal is to find such partitioning of the graph nodes that minimizes the number of disagreements on the edges. They concluded that the problem is NP-hard and a provided constant factor approximation algorithm for it. They also introduced an extended problem called weighted correlation clustering. Here, each edge is assigned a positive weight indicating the importance of labeling it correctly.

Demaine *et al.* (2006) studied weighted correlation clustering on general graphs which is equivalent to (MC). A very important contribution of this work is the proof that the weighted correlation clustering and therefore also multicut problem is equivalent to the classical multicut problem. This finding thus bridges the gap between the two different multicut Definitions 2.1 and 2.2.

Several authors study complexity and approximation algorithms for (weighted) correlation clustering (Bansal *et al.*, 2004; Demaine *et al.*, 2006; Charikar *et al.*, 2005). However, the optimal values of the multicut problem and weighted correlation clustering differ by a constant. Therefore, approximation guarantees for the weighted correlation clustering cannot be transferred to the multicut.

Coalition structure. The optimal coalition structure in a weighted graph game (Bachrach *et al.*, 2013) coincides with (MC) on the given graph. In contrast to (MC), the aim is to maximize the sum of edge weights within each partition (coalition). Therefore, the objective values of the two tasks differ by a constant and the approximation results are not directly transferable between the two tasks. Some complexity and approximation results for this problem can be found in Voice *et al.* (2012) and Bachrach *et al.* (2013).

2.1.2 ALGORITHMS FOR SOLVING MULTICUT

Branch and cut. Branch and cut is a method for solving ILP optimally. If stopped earlier, it can usually provide an approximate solution together with its gap to the optimum. There exist commercial solvers that implement branch and cut efficiently. Since there is an exponential number of multicut cycle inequalities 2.1, they are typically added to the problem in a cutting plane method. That is, a cycle inequality is added whenever the solver outputs a candidate solution that violates it. Andres *et al.* (2011) applied this method on solving multicut for image segmentation and Andres *et al.* (2012) for decomposing connectomics data. Pishchulin *et al.* (2016) and Insafutdinov *et al.* (2016) used a branch and cut multicut solver for multi-person pose estimation. Despite reasonable efficiency on some real data, branch and cut can take exponential time in general. This fact typically prohibits its usage on large data.

Cutting plane for LP relaxation. Many approaches use cutting plane methods for solving linear programming (LP) relaxation of (MC). Such algorithms iteratively output the optimal solution with respect to a subset of valid inequalities. After that, separation procedures are used to identify constraints violated by the solution and add them to the constraint set. This approach is repeated until a solution that does not violate any given constraints is found. The resulting solution is often not integral. Therefore, it is rounded to an integral one using a heuristic. Such solutions are not guaranteed to be optimal. However, the optimal solution of the relaxation gives a lower bound of the optimal ILP solution. Grötschel and Wakabayashi (1989) used the cutting plane for multicuts on complete graphs. Their solution satisfies all triangle inequalities and they use heuristics to separate some other inequalities. Nowozin and Jegelka (2009) used a cutting plane method that either uses only cycle inequalities or includes also odd-wheel inequalities. They separate odd-wheel inequalities via a polynomial-time procedure from Deza *et al.* (1992). One variant of the method suggested by Kappes *et al.* (2011) switches to an exact ILP solver by adding integrality constraints instead of using a rounding heuristic and thus outputs the optimal solution. Kappes *et al.* (2016) compares several versions of multicut cutting plane algorithms including algorithms for higher-order multicut (multicut on hypergraphs). Their methods were also used by Kappes *et al.* (2015b). Kim *et al.* (2011) and Kim *et al.* (2014) employed cutting plane algorithms for solving higher order multicut.

Lagrangean decomposition. Another approach to solve multicut is to use Lagrangean (dual) decomposition. We provide details about this method in Section 5.3. In this case, the combinatorial problem is decomposed into smaller but feasible subproblems that are optimized independently and so provide a lower bound to the primal problem. The goal is to maximize this lower bound. Unfortunately, there is typically no clear way how to obtain a primal feasible solution from the dual one. Therefore, it is necessary to use heuristics. Various versions of Lagrangian decomposition for multicut were employed by Yarkony *et al.* (2012), Yarkony *et al.* (2014), and Andres *et al.* (2013). Swoboda *et al.* (2017a) designed a message passing framework that monotonically improves the lower bound given by the Lagrangean decomposition of a combinatorial problem. Swoboda and Andres (2017) developed multicut-specific methods to be used within this framework. Abbas and Swoboda (2021) proposed a primal-dual algorithm for multicut computation. In each iteration, it performs parallel message passing followed by a parallel edge contraction. The contraction is based on maximum matching of reparametrized edge costs obtained from the message passing. These two steps are repeated until no edge contraction candidates can be found anymore. As a result, the algorithm provided high-quality solutions together with lower bounds significantly faster than previous methods.

Heuristic solutions. Local search heuristics have proven to be efficient methods for obtaining multicut solutions. Although they do not provide any guarantees with respect to the gap to the optimum, they often perform well in practice. They work with feasible solutions that are iteratively improved by local moves. Beier *et al.* (2014) developed a method called Cut, glue & cut and Beier *et al.* (2015) developed a fusion move method. Levinkov *et al.* (2017a) provided a comparison of some multicut heuristics on instances like 2D and 3D image segmentation and clustering of handwritten digits. They compared the method from Beier *et al.* (2014), greedy additive edge contraction (GAEC), greedy fixations (GF), and several implementations Kernighan Lin (KL) heuristic (Kernighan and Lin, 1970), namely those from Nowozin and Jegelka (2009), Kappes *et al.* (2015a) and a modification called Kernighan Lin with joins (KLj) proposed by Keuper *et al.* (2015b) for solving lifted multicut problems. KL heuristic for solving multicut was also adopted by Keuper *et al.* (2015a), Levinkov *et al.* (2016), Tang *et al.* (2015), and Tang *et al.* (2016).

Hierarchical solutions. When dealing with big problems, it can be time-consuming to solve the whole instance at once even with heuristics. Therefore, many researchers developed methods for solving multicut for subproblems and combining the local solutions into consistent partitioning of the whole instance. Kroeger *et al.* (2013) proposed a block scheme for solving multicut instances for connectomics. Pape *et al.* (2017) designed a hierarchical block-wise scheme where subproblems can be solved in parallel. Ho *et al.* (2021) proposed a method called multi-stage multicuts (MSM) where multiple CPU threads use a shared memory system. Multicut is solved on subsets of the whole problem of the same size which is followed by a merge process.

Persistency criteria. A useful tool for reducing the size of a combinatorial problem is studying persistency, that is partial variable assignments that agree with some optimal solution. Alush and Goldberger (2012) proposed two partial optimality conditions for (MC). Lange *et al.* (2018) studied the persistency criteria for (MC) and max-cut and proposed routines and fast algorithms to verify them. The methods can be used as preprocessing techniques that reduce problem sizes if variables are fixed to their persistent values. The authors also proposed an algorithm called iterative cycle packing that provides a dual lower bound and values for graph edges that can be used for their re-weighting. Applying known heuristics (GAEC and KLj) to problems with re-weighted edge costs yielded better results than applying the heuristics on the graph with the original edge costs. Lange *et al.* (2019) proposed more involved persistency criteria and methods to verify them. Their application substantially reduced the number of variables in problem instances from various multicut datasets.

2.1.3 APPLICATIONS OF MULTICUT IN COMPUTER VISION

Image segmentation. A common computer vision application of multicut is image segmentation for which (MC) is a convenient model. Here, the image can be represented by an adjacency graph where the nodes usually represent image superpixels. The following authors applied multicut to image segmentation: Andres *et al.* (2011), Beier *et al.* (2014), Beier *et al.* (2015), Kim *et al.* (2014), Kappes *et al.* (2011), Kappes *et al.* (2015b), Yarkony *et al.* (2012), Yarkony *et al.* (2014), Kardoost and Keuper (2021) and Alush and Goldberger (2012). Formulating an improved model for image segmentation motivated Kim *et al.* (2011) and Kappes *et al.* (2016) to define and study higher order correlation clustering, i.e. correlation clustering on hypergraphs. Abbas and Swoboda (2021) provided a fast parallel algorithm enabling to solve (MC) over image pixels of high-resolution images.

Segmenting electron microscopy data. Multicut turned out to be very useful for segmenting cells in 2D and 3D data from electron microscopy (Zhang *et al.*, 2014; Wolny *et al.*, 2020). 3D segmentation of neural circuits (connectomics) was done by Andres *et al.* (2012), Kroeger *et al.* (2013), Beier *et al.* (2014), Beier *et al.* (2015), Pape *et al.* (2017) and Abbas and Swoboda (2021).

User-assisted image segmentation. Some software tools enable to perform multicut-based image segmentation that satisfies constraints provided by the user (Andres *et al.*, 2013; Levinkov *et al.*, 2016)). Enabling this user input can help to correctly segment biological data. This fact is used in an interactive machine learning tool for biological image analysis presented by Berg *et al.* (2019) where multicut is an integral part of some of the underlying methods.

Joint labeling and decomposition. By extending the multicut model by variables for node labeling, it is possible to perform image segmentation and labeling jointly (Kappes *et al.*, 2011). Kirillov *et al.* (2017) used such a model for solving the task of instance aware semantic segmentation.

Multiple object tracking. (MC) turned out to be a suitable model for multiple object tracking. In Tang *et al.* (2015) and Tang *et al.* (2016), person detections are represented by graph vertices and the task is to group together those detections that belong to the same person. Alternatively, graph vertices can represent tracklets (Ristani and Tomasi, 2014).

Video and motion segmentation. Besides the target application in connectomics data, Andres *et al.* (2012) also used their multicut solver for video segmentation. Keuper *et al.* (2015a) used multicut for obtaining point trajectories in a video for solving the task of motion segmentation. Keuper *et al.* (2018) used a combination of two multicuts on two graphs (on object detections and point trajectories) and applied the resulting model called correlation co-clustering on solving the task of motion segmentation and multiple object tracking together. Kardoost and Keuper (2021) included uncertainty in their multicut-based motion segmentation method.

Multi-Person Pose Estimation and Tracking. Pishchulin *et al.* (2016) worked with body parts detections and introduced a multicut model extended by variables for labeling these detections and connecting them into poses of people. Insafutdinov *et al.* (2016) improved the method from Pishchulin *et al.* (2016). Song *et al.* (2019) explored a different approach for multi-person pose estimation. They formulated the multicut problem on a complete graph as an unconstrained binary cubic problem with a high positive cost of inconsistent triangle labeling. Their formulation enabled to learn variable costs end-to-end for the desired applications. Insafutdinov *et al.* (2017) used a simpler multicut model where the body parts detections take only binary labels indicating whether they are true or false detections. This model was originally introduced for MOT by Tang *et al.* (2015). Insafutdinov *et al.* (2017) applied the method to both multiple people pose estimation in a single image and tracking multiple people poses in videos.

Clustering. The applications of (MC) comprise many clustering problems on both complete and non-complete graphs. The domains of interests are for instance social networks, clustering of objects like animal species based on a set of characteristics, clustering of images, etc. These applications were inspected e.g. by Grötschel and

Wakabayashi (1989), Ho *et al.* (2021), Song *et al.* (2019), Beier *et al.* (2015).

2.2 LIFTED MULTICUT PROBLEM

We first define the minimum cost lifted multicut problem (LMC). More details can be found in Section 3.3.

In the case of multicut of graph $G = (V, E)$, the cost of an edge expresses the preference of whether its endpoints should be connected or not. Lifted multicut enables to impose connectivity priors on arbitrary pairs of nodes and not only those connected by an edge in G. Given graph $G = (V, E)$, we are interested in lifted graph $G' = (V, E')$ where $E \subseteq E' \subseteq \binom{V}{2}$. If there is a lifted edge between two nodes, its cost provides the preference whether the nodes should belong to the same component of graph G or not. That is, we deal with decompositions of graph G represented by edges in E' that straddle distinct components (Figure 2.2).

Let us have $M \subseteq E'$. Then M is a *multicut of G' lifted from G* if and only if the two following conditions hold. First, $M \cap E$ is a multicut of G. Second, for each edge $vw \in E' \setminus E$ it holds that vw belongs to M if and only if v and w are in distinct components of G as defined by $M \cap E$.

Similarly as for multicut, we introduce *characteristic functions of lifted multicut* $x \in \{0, 1\}^{E'}$. By the means of the characteristic function, the condition on lifted edges reads,

$$\forall vw \in M \setminus E : \ (x_{vw} = 1 \Leftrightarrow \forall P \in vw\text{-paths}(G) : \ \exists ij \in P : x_{ij} = 1) \, . \tag{2.2}$$

We denote by $X_{GG'}$ the set of all characteristic functions of multicuts of G' lifted from G. That is, $X_{GG'} = \{x \in \{0, 1\}^{E'} | x \text{ satisfies } (2.1) \wedge (2.2)\}$.

We provide an exact definition of $X_{GG'}$ through linear inequalities in Section 3.3.2.

Definition 2.3. (LMC) *Given graph $G = (V, E)$ and lifted graph $G' = (V, E')$ where $E \subseteq E' \subseteq \binom{V}{2}$ and a cost function $c : E' \to \mathbb{R}$, the 0/1 linear program written below is called an instance of the* Minimum Cost Lifted Multicut Problem *w.r.t. G, G', and c.*

$$\min_{x \in X_{GG'}} \sum_{e \in E'} c_e x_e \tag{LMC}$$

The formulation of the lifted multicut problem was motivated by solving the task of image segmentation (Keuper *et al.*, 2015b). On the other hand, our work Horňáková *et al.* (2017) presented a comprehensive study of the theoretical properties of this model.

Related problems. For several combinatorial problems, special connectivity-inducing edges have been introduced to improve the expressiveness of the base problem. In the Markov Random Field literature, the problem was studied from a polyhedral point of view by Nowozin and Lampert (2010). The special edges were used in the image analysis to indicate that two non-adjacent pixels come from the same object and hence they must belong to a contiguously labeled component of the underlying graph.

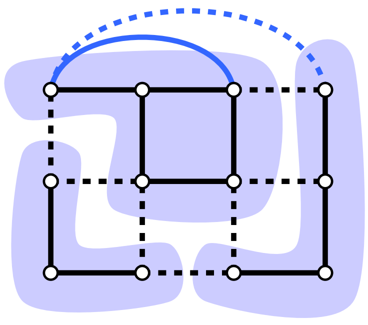


Figure 2.2: An example of multicut of G' lifted from G. The decomposition of graph G (black edges) is characterized by the set of edges of G' that straddle distinct components (dashed lines). Lifted edges are depicted in blue.

Theoretical study. Our work Horňáková *et al.* (2017) provided the definition of the lifted multicut problem and studied it from several perspectives. Among its highlights is proving that the dimension of the lifted multicut polytope is equal to the number of lifted graph edges. It also provided some criteria for facet defining inequalities. Other important contributions were exploring the possibility of partial edge labeling and defining general metrics for graph decompositions based on partial information. A substantial part of this work is contained in Chapter 3. Lange and Andres (2020) studied the case when the base graph is a tree. They discovered that the problem stays NP-hard in general but it is polynomially solvable for paths. They characterized facets of the lifted multicut polytope for trees. The facets represent a complete description of the lifted multicut polytope for paths.

2.2.1 ALGORITHMS FOR SOLVING LIFTED MULTICUT

Branch and cut. We presented in our paper Horňáková *et al.* (2017) a branch and cut procedure for solving (LMC) optimally. The implementation is publicly available at https://github.com/bjoern-andres/graph.

Heuristics. Although the branch and cut implementation for obtaining the optimal solution of (LMC) exists, most of the applications used heuristics for its solution. Keuper *et al.* (2015b) proposed a modified version of KL heuristic called Karnighan Lin with joins (KLj) which has been adopted as a solver for both multicut and lifted multicut problems by many other authors (Keuper *et al.*, 2018; Tang *et al.*, 2016, 2017; Song *et al.*, 2019). They obtain an initial segmentation from GAEC and use KLj to improve it. Levinkov *et al.* (2017b) presented two variants of KLj extended for the task of joint graph decomposition and node labeling. KL heuristic was also used by Keuper (2017) who extended the algorithm to work with higher order lifted graphs. Beier *et al.* (2016) extended the fusion move algorithm proposed by Beier *et al.* (2015) to solve (LMC). Kardoost and Keuper (2019) used two modified versions of GAEC where the order of edges to be merged is modified.

Hierarchical solutions. Similarly as for multicut, it is often necessary to decompose big instances into smaller subproblems and define a merging strategy for combining their solutions. Pape *et al.* (2019) extended their hierarchical solver Pape *et al.* (2017) to solve (LMC).

2.2.2 APPLICATIONS OF LIFTED MULTICUT IN COMPUTER VISION

Image segmentation and 3D decomposition. The possibility to include connectivity preferences about non-neighboring superpixels and thus using non-planar graphs for image segmentation was inspected by Andres *et al.* (2013). This extension can be viewed as the first step towards introducing lifted edges and thus extending multicut to lifted multicut. For a pair of neighboring pixels or superpixels in a real image, it is often hard to estimate whether the image should be cut precisely between these pixels. On the other hand, it is often easy to estimate for pixels at a larger distance whether they belong to distinct objects. Therefore, Keuper *et al.* (2015b) proposed the lifted multicut problem and applied it to the tasks of image segmentation and mesh decomposition. Further authors that used the lifted multicut problem for image segmentation and 3D decomposition were Kardoost and Keuper (2019) and Beier *et al.* (2016) who also inspected averaging of multiple segmentations.

Segmenting electron microscopy data. Beier *et al.* (2016) and Beier *et al.* (2017) used lifted multicut for segmenting 3D electron microscopy images of a brain. Pape *et al.* (2019) applied lifted multicut to several types of 3D microscopy data.

Joint labeling and decomposition. The authors of Levinkov *et al.* (2017b) introduced a joint task of lifted multicut and node labeling and applied their model to articulated human body pose estimation and instance separating semantic segmentation.

Multiple object tracking. A variant of (MC) was successfully applied to multiple object tracking. Therefore, after introducing (LMC), a natural question arose whether it can advance the tracking performance. In particular, can information about connectivity preferences of detections that are far away in time improve the tracking accuracy? Based on this fact, Tang *et al.* (2017) successfully employed (LMC) into their tracking system and showed that (LMC) based tracking achieves better performance than the one using only multicut. Other authors employing lifted multicut for MOT were Ho *et al.* (2020) and Levinkov *et al.* (2017b).

Motion segmentation. Applying multicut to obtaining point trajectories for motion segmentation has proven to be useful by Keuper *et al.* (2015a). Keuper (2017) solved this task via extending the previous method to the higher order lifted multicut.

2.3 DISJOINT PATHS PROBLEM.

We start with the basic theory of the disjoint paths problem. We consider directed graphs in this work. Analogical definitions exist for the undirected case too. Disjoint paths problem in a directed graph is a special case of the network flow problem when the flow through each edge is restricted to be binary. If the paths are required to be node-disjoint, the flow through each node that is not a terminal is restricted to be binary too. Consequently, we can use a restricted definition of the feasible network flow between *source node* s and *sink node* t to define the set of edge-disjoint paths between the two nodes.

Definition 2.4. *Let $G = (V, E)$ be a directed graph with two special nodes source s and sink t given. We denote by $Y^{EDP}(G, s, t)$ the set of all $y \in \{0, 1\}^E$ that satisfy the following constraints*

$$\forall v \in V \setminus \{s, t\} : \quad \sum_{uv \in E} y_{uv} - \sum_{vw \in E} y_{vw} = 0\,, \tag{2.3}$$

$$\sum_{sv \in E} y_{sv} \geq 0\,. \tag{2.4}$$

Here, (2.3) are flow conservation constraints and (2.4) requires that the flow originates in s. Moreover, constraints (2.3) ensure that the flow originating in s is the same as the flow absorbed by t. Due to the requirement that y_{uv} is binary for each edge $uv \in E$, each vector $y \in Y^{EDP}(G, s, t)$ represents a set of edge-disjoint st-paths in G. We will sometimes write only Y^{EDP} for brevity.

In this work, we concentrate on node-disjoint paths. They are defined similarly.

Definition 2.5. *Let $G = (V, E)$ be a directed graph with two special nodes source s and sink t given. We denote by $Y^{DP}(G, s, t)$ the set of all $y \in \{0, 1\}^E$ that satisfy (2.3), (2.4) and the following constraints*

$$\forall v \in V \setminus \{s, t\} : \quad \sum_{uv \in E} y_{uv} \leq 1\,. \tag{2.5}$$

It can be considered that every $y \in Y^{DP}(G, s, t)$ represents a set of node-disjoint st-paths in G. Clearly, Constraint (2.5) ensures that there is maximally one path going through each node (except for s and t). We will again often write only Y^{DP} for brevity.

Network flow. Many publications explore flow in networks and various tasks related to it. One of the first studies is Ford and Fulkerson (1956) that formulated the famous min-cut max-flow theorem. Other pioneering work is for instance Ford Jr (1956) and Dantzig and Fulkerson (1955). For a comprehensive overview of results from the network flow theory, we refer e.g. to the books Ahuja *et al.* (1988) and Schrijver (2003).

2.3.1 THE MAXIMUM NUMBER OF DISJOINT PATHS.

Using the notation from Definition 2.5, we can formally define the problem of finding the maximum number of node-disjoint and edge-disjoint st-paths in a graph.

Definition 2.6. *Given a directed graph $G = (V, E)$ and two nodes $s, t \in V$ the task of finding* a maximum number of node-disjoint st-paths *is*

$$\max_{y \in Y^{DP}(G,s,t)} \sum_{sv \in E} y_{sv} . \tag{$\mathsf{DP}_{\mathsf{max}}$}$$

Definition 2.7. *Given a directed graph $G = (V, E)$ and two nodes $s, t \in V$ the task of finding* a maximum number of edge-disjoint st-paths *is*

$$\max_{y \in Y^{EDP}(G,s,t)} \sum_{sv \in E} y_{sv} . \tag{$\mathsf{EDP}_{\mathsf{max}}$}$$

The first result about finding a set of disjoint paths between nodes s and t in a graph is due to Menger (1927). One of the results of this work directly implies the following theorem which we state in the form provided by Schrijver (2003).

Theorem 2.1. *(Menger's theorem). Let $G = (V, E)$ be a digraph and let $S, T \subseteq V$. Then the maximum number of node-disjoint ST-paths is equal to the minimum size of an ST-disconnecting node set.*

This theorem is a special version of min-cut max-flow theorem (Ford and Fulkerson, 1956). Variants of Menger's theorem exist for undirected graphs (Grünwald, 1938) and also for edge-disjoint paths in directed (Dantzig and Fulkerson, 1956) and undirected graphs (Kotzig, 1956). In the edge-disjoint version, the minimum size st-cut replaces the node set. Obviously, finding the disjoint paths between sets $S, T \subseteq V$ is equivalent to finding a set of disjoint paths between nodes $s, t \in V$. We can for instance add nodes s and t to the graph G and add edges from s to all nodes in set S and add edges from all nodes in T to node t. Conversely, we can set $S := \{v \in V | sv \in E\}$ and $T := \{v \in V | vt \in E\}$.

Algorithms for finding the maximum number of disjoint paths. A substantial effort has been devoted to the development of an efficient algorithm for finding the maximum number of st-paths in all variants for directed and undirected graphs and for node-disjoint as well as edge-disjoint version. For the edge-disjoint version, Ford and Fulkerson (1957) gave an algorithm having complexity $\mathcal{O}(|E|^2)$. More efficient algorithms were given e.g. by Karzanov (1973), Tarjan (1974), Even and Tarjan (1975) and Nagamochi and Ibaraki (1992). For the node-disjoint version, efficient algorithms were given by Karzanov (1973), Tarjan (1974), Even and Tarjan (1975), Nagamochi and Ibaraki (1992) and Feder and Motwani (1991). For planar graphs, algorithms linear in the number of vertices exist for various variants (Weihe, 1997; Brandes and Wagner, 2000; Ripphausen-Lipa *et al.*, 1997). See Schrijver (2003) for more details.

2.3.2 DISJOINT PATHS BETWEEN SPECIFIED PAIRS OF NODES

In the previous section, we considered the problem of finding the maximum number of the node-disjoint or edge-disjoint paths between one source node s and one sink node t which is a binary version of the maximum flow of a single commodity. In this section, we will concentrate on finding k disjoint paths such that for each path a specific pair of source and sink nodes is given. That is, we are looking for a set of paths $P_1, \ldots, P_k$ such that the path P_i starts in s_i and terminates in t_i. This problem is studied for both directed and undirected graphs. However, we again concentrate only on directed graphs because they are more relevant to our work.

Notation. We call the graph $G = (V, E)$ where the disjoint paths are searched the *base graph*. This is in compliance with our Chapters 4 and 5 and also with e.g. Oellrich (2008). This graph is also denoted as the *supply graph* in the literature (e.g. in Vygen (1994)). The pairs of source-sink nodes are typically represented by a graph $H = (T, D)$ called the *demand graph* where $T \subseteq V$ and $D = \{(s_1, t_1), \ldots, (s_k, t_k)\}$. The nodes in H are called *terminals*. H may also contain parallel edges (i.e. multiple edges between one pair of nodes). We denote by $\tilde{H} = (T, \tilde{D})$ the demand graphs where parallel edges between the same terminals having the same orientation are replaced by one edge between the two terminals.

Definition 2.8. *Given a directed graph $G = (V, E)$ and a directed demand graph $H = (T, D)$ where $|D| = k$, the solution of the* directed edge-disjoint paths problem *w.r.t. G and H is a collection of edge-disjoint paths $(P_1, \ldots, P_k)$ such that $\forall i = 1, \ldots, k :$ $P_i \in s_it_i$-paths(G).*

Definition 2.9. *Given a directed graph $G = (V, E)$ and a directed demand graph $H = (T, D)$ where $|D| = k$, the solution of the* directed node-disjoint paths problem *w.r.t. G and H is a collection of paths $(P_1, \ldots, P_k)$ such that $\forall i = 1, \ldots, k : P_i \in s_it_i$-paths$(G)$ and the paths are mutually node disjoint up to the terminals.*

Relation to multi-commodity flow. We can view the edge-disjoint paths problem w.r.t. G and H as the integer multi-commodity flow where the i-th edge $(s_i, t_i) \in \tilde{D}$ corresponds to commodity i, the number of edges in D parallel with (s_i, t_i) are equal to the requirement R_i of commodity i and the capacity of each edge is equal to one.

Relation to lifted disjoint paths. This problem is relevant for our lifted disjoint paths problem because solving (LDP) is at least as hard as finding a set of node-disjoint paths in a directed acyclic graph. This is NP-complete in general as we describe later in this section. We provide details of the respective transformation in Section 4.6.1.

Transformations between node-disjoint and edge-disjoint paths problems. LaPaugh and Rivest (1980) provided polynomial-time transformations from the node-disjoint paths problem to the edge-disjoint paths problem and vice versa. Both transformations keep the number of demand edges $|D|$ as well as the number of edges in the simplified demand graph $|\tilde{D}|$. Moreover, the transformation of an acyclic graph results into an acyclic graph in both cases. These results enable to generalize NP-completeness results for one task to the other task.

General Complexity. Unlike in the previous problems ($\mathsf{DP_{max}}$) and ($\mathsf{EDP_{max}}$) where a single source-sink pair is given for all paths, the problem with multiple source-sink pairs is NP-complete in general for both node-disjoint and edge-disjoint paths (Karp, 1975). Below, we provide some special cases where the problems remain NP-complete and where they are polynomially solvable.

NP-complete special cases. We provide the basic NP-completeness results in the form as summarized in two theorems by Oellrich (2008). We refer to Oellrich (2008) for more details. Some of the results follow from the stated publications after applying the above mentioned transformations between the node-disjoint and edge-disjoint versions.

Theorem 2.2. *Given graph $G = (V, E)$ and the demand graph $H = (T, D)$, deciding whether the directed node-disjoint paths problem w.r.t. G and H has a solution is NP-complete*

1. *in general (Karp, 1975),*
2. *even if $|D| = 2$ (Fortune et al., 1980),*
3. *even if G is acyclic and $|D| = 2$ (Even et al., 1975),*
4. *even if $(V, E \cup D)$ is planar (Vygen, 1995).*

Theorem 2.3. *Given graph $G = (V, E)$ and the demand graph $H = (T, D)$, deciding whether the directed edge-disjoint paths problem w.r.t. G and H has a solution is NP-complete*

1. *in general (Karp, 1975),*
2. *even if $|D| = 2$ (Fortune et al., 1980),*
3. *even if G is acyclic and $|D| = 2$ (Even et al., 1975),*
4. *even if G is planar, $(V, E \cup D)$ is Eulerian and $|D| \leq 3$ Vygen (1995),*
5. *even if $(V, E \cup D)$ is planar (Vygen, 1995),*
6. *even if G is planar and acyclic (Vygen, 1995).*

Polynomially solvable cases. There are some special cases of the k-disjoint paths problem that are solvable in polynomial time. For instance, the problems ($\mathsf{DP_{max}}$) and ($\mathsf{EDP_{max}}$) described in the previous section correspond to the case when $|\tilde{D}| = 1$ and we already know that they are polynomially solvable. Tholey (2012) proposed an efficient method for directed acyclic graphs when $k = 2$. He also provided a good overview of efficient methods for other special variants of the tasks discussed in this section. Polynomially solvable cases with a fixed number of paths are described in the two following theorems (provided in the form by Oellrich (2008)). Again, for obtaining some of the results, one needs to apply the transformations proposed by LaPaugh and Rivest (1980).

Theorem 2.4. *Let $k \in \mathbb{N}$ be fixed. Given graph $G = (V, E)$ and the demand graph $H = (T, D)$. There exists a polynomial-time algorithm for the directed node-disjoint paths problem w.r.t. G and H if*

1. G is acyclic and $|D| \leq k$ (Fortune et al., 1980),

2. G is planar and $|D| \leq k$ (Schrijver, 1994).

Theorem 2.5. *Let $k \in \mathbb{N}$ be fixed. Given graph $G = (V, E)$ and the demand graph $H = (T, D)$. There exists a polynomial-time algorithm for the directed edge-disjoint paths problem w.r.t. G and H if*

1. G is acyclic and $|D| \leq k$ (Fortune et al., 1980),

2. $(V, E \cup D)$ is Eulerian and $|D| = 3$ (Ibaraki and Poljak, 1991).

Shortest disjoint paths between specified pairs of nodes. Eilam-Tzoreff (1998) studied the problem of finding the shortest disjoint paths in the case that k specific source-sink pairs are given. He concludes that the problem is NP-complete in all four variants, i.e. for both for directed and undirected graphs and for the node-disjoint and edge-disjoint variant.

2.3.3 MINIMUM COST DISJOINT PATHS

The task of finding the k minimum cost node-disjoint or edge-disjoint paths between nodes s and t is a special case of minimum cost flow problem (see e.g. Ahuja *et al.* (1988)). Similarly as for the previous disjoint paths problems, the edge capacity and eventually the node capacity is one and the flow through each edge (and node) is restricted to be binary. The required number of paths k corresponds to the flow requirement.

Minimum cost flow problem. We start with some basic definitions and theory according to Schrijver (2003).

Definition 2.10. *Let $G = (V, E)$ be a digraph and let $c : E \to \mathbb{R}$, called the* cost function*. For any function $y : E \to \mathbb{R}$, the cost of y is*

$$\sum_{e \in E} c_e y_e \tag{2.6}$$

Definition 2.11. *Given a digraph $G = (V, E)$, terminals $s, t \in V$, capacity function $u : E \to \mathbb{Q}^+$, cost function $c : E \to \mathbb{Q}$ and flow requirement R,* minimum cost st-flow problem (MCF) *is to find an st-flow y of value R that satisfies the capacity constraint on each edge, i.e. $\forall e \in E : y_e \leq u_e$ and minimizes cost (2.6) of y.*

Definition 2.12. *Let us have a digraph $G = (V, E)$, terminals s, t, and a feasible st-flow $y : E \to \mathbb{R}$. We define* flow through nodes *of G w.r.t. y as a mapping $z : V \setminus \{s, t\} \to \mathbb{R}$ where*

$$\forall v \in V \setminus \{s,t\} : \quad z_v = \sum_{uv \in E} y_{uv} \,. \tag{2.7}$$

Integral solutions. The following theorem states a known property of minimum cost flow with integral capacities, see e.g. Ahuja *et al.* (1988).

Theorem 2.6. *If all edge capacities and supplies/demands of nodes are integers, the minimum cost flow problem always has an integer minimum cost flow.*

Proof can be found e.g. in Ahuja *et al.* (1988). It follows from the fact that the minimum cost flow problem can be stated as a linear program in the form $\{\min cx | Ax \geq b, x \leq 0\}$ where A is a totally unimodular matrix. It holds that if b is integral, the program has an optimal integral solution (Hoffman and Kruskal, 1956).

Minimum cost flow algorithms. Minimum cost flow problem can be solved in polynomial time e.g. by successive shortest paths algorithm. The algorithm was developed independently by Jewell (1958), Iri (1960), and Busacker and Gowen (1960) and improved by Edmonds and Karp (1972) and Tomizawa (1971). Kovács (2015) provided an extensive computational analysis of several algorithms for solving (MCF) and also his own implementation of some of the algorithms.

Minimum cost k disjoint paths algorithm. The algorithms for solving the (MCF) problem can be also used for finding minimum cost disjoint paths thank to Theorem 2.6. Suurballe (1974) presented an algorithm for finding k shortest node-disjoint paths in a weighted directed graph. The k paths are found in k iterations of a single shortest path algorithm. In particular, in the i-th iteration, i shortest paths are found inductively from the optimal solution for the $i-1$ shortest paths.

Minimum cost disjoint paths with arbitrary k. The target application of our lifted disjoint paths problem is multiple object tracking (MOT). Therefore, we are interested in applications of disjoint paths in MOT. In contrast to the problem solved by Suurballe (1974), the number of disjoint paths in MOT is typically not known in advance. Therefore, it is advantageous to determine it purely from the edge costs, similarly to the number of clusters in the minimum cost multicut problem. In particular, a negative cost of an edge indicates that it is preferable to have a flow through that edge and a positive edge cost indicates the opposite (see Zhang *et al.* (2008)). For MOT, we can restrict ourselves to directed acyclic graphs where each edge leads from an object detection in time T to an object detection in time T' where $T' > T$. We also allow node costs in our model. Note that such a model can be easily transformed to an edge-disjoint path problem with cost function only on edges if each node is replaced by an additional edge. For our purposes, the minimum cost node-disjoint paths problem is the task from Definition 2.13.

Definition 2.13. (DP) *Let us have a directed acyclic graph $G = (V, E)$, source node s, sink node t, an edge cost function $c : E \to \mathbb{R}$ and a node cost function $d : V \to \mathbb{R}$. Let all vertices $v \in V$ be reachable from s and let t be reachable from all vertices $v \in V$. We define the* minimum cost node-disjoint paths *problem as*

$$\min_{\substack{y \in \{0,1\}^E \\ z \in \{0,1\}^V}} \sum_{e \in E} c_e y_e + \sum_{v \in v} d_v z_v \qquad s.t. \tag{DP}$$

$$y \in Y^{DP}(G, s, t) \ \wedge \ z \text{ satisfies } (2.7)$$

Recall that set $Y^{DP}(G, s, t)$ requires binary flow through nodes and edges and flow conservation in nodes (2.3). However, the number of paths (flow requirement) is not given.

Minimum cost disjoint paths in MOT. Many MOT systems employ (DP) while using various approaches to determine the optimal number of paths. Zhang *et al.* (2008) used the fact that the minimum cost flow is a convex function w.r.t. the required amount of flow. Therefore, they suggested using the Fibonacci search over the possible number of paths k. In each step, they performed one execution of the min-cost flow algorithm by Goldberg (1997). Ma *et al.* (2009) used the algorithm proposed by Suurballe (1974) for each k within an interval $k_l \leq k \leq k_u$.

In fact, the optimal number of paths can be determined much simpler in one call of the successive shortest paths algorithm resp. the algorithm from Suurballe (1974). Berclaz *et al.* (2011) ran the algorithm from Suurballe (1974) only once and simply returned the solution found in the k-th iteration if the $k + 1$ iteration lead to an increase of the global cost. Similarly, Pirsiavash *et al.* (2011) suggested finding the globally optimal solution using one call of the successive shortest paths algorithm. In this case, unit flow is added in each iteration which practically increases the number of paths by one. Again, the search terminates if the next iteration increases the global cost. Wang *et al.* (2019a) proposed an algorithm called minimum-update successive shortest path which performs extremely well in practice.

Leal-Taixé *et al.* (2011) solved (DP) using an efficient implementation of the Simplex algorithm (Dantzig, 2016). Other MOT methods employing (DP) comprise Leal-Taixé *et al.* (2014) and Braso and Leal-Taixe (2020).

ANALYSIS OF GRAPH DECOMPOSITIONS BY LIFTED MULTICUTS 3

3.1 INTRODUCTION

We study the set of all decompositions (clusterings) of a graph represented by the multicut and the lifted multicut. This chapter is based on our work Horňáková *et al.* (2017). We start with a theoretical introduction to the multicut and lifted multicut problem in Section 3.3 which is more detailed than the theory in Chapter 2.

After that, we study the lifted multicut in three ways. First, we study classes of decompositions (partial lifted multicuts) represented by *must-cut* and *must-join* constraints on a subset of (lifted) edges and problems arising from this representation (Section 3.4). Such constraints have applications in domains where defining a complete decomposition is an ambiguous and tedious task, e.g. in the field of image segmentation. The first arising problem is to decide whether a set of must-join and must-cut constraints is *consistent*, i.e., whether a decomposition of the given graph exists that satisfies the constraints. We show that this decision problem is NP-complete in general and can be solved efficiently for a subclass of constraints. The second problem is to decide whether a consistent set of must-join and must-cut constraints is *maximally specific*, i.e., whether no such constraint can be added without changing the set of decompositions that satisfy the constraints. We show that this decision problem is NP-hard in general and can be solved efficiently for a subclass of constraints. This finding is relevant for comparing the classes of decompositions definable by must-join and must-cut constraints by certain metrics, which is the next topic.

As the second application of lifted multicuts, we study the comparison of decompositions and classes of decompositions by *metrics* (Section 3.5). To obtain a metric on the set of all decompositions of a given graph, we define a metric on a set of lifted multicuts that characterize these decompositions. By lifting to different graphs, we obtain different metrics, two of which are well-known and here generalized. To extend this metric to the classes of decompositions definable by must-join and must-cut constraints, we define a metric on partial lifted multicuts that characterize these classes, connecting results of Sections 3.4 and 3.5. We show that computing this metric is NP-hard in general and efficient for a subclass of must-join and must-cut constraints. These findings have implications on the applicability of must-join and must-cut constraints as a form of supervision, more specifically, on the practicality of certain error metrics and loss functions.

As the third application of lifted multicuts, we study the lifted multicut polytope. We study its dimension (Section 3.6.1) and the properties of some of its facets (Figure 3.1

and 3.2). However, the respective Section 3.6 does not contain all findings provided in Horňáková *et al.* (2017). Therefore, we refer to the full paper for further results.

3.2 RELATED WORK

We provide an overview of the work related to the multicut and lifted multicut problems, their applications and algorithms in Chapter 2.

The must-join and must-cut constraints (also called cannot-link and must-link) as hard constraints for clustering were introduced by Wagstaff and Cardie (2000). They incorporated a few randomly generated constraints into a known clustering algorithm COBWEB (Fisher, 1987) and observed an improved performance. Davidson and Ravi (2005) explored the feasibility of satisfying all clustering constraints when a lower bound and an upper bound on the number K of clusters is given. Besides the must-join and must-cut constraints, they also introduced δ-constraints as the thresholds on the minimal distance between two points in different clusters and ϵ-constrains as the maximal distance to the nearest neighbor belonging to the same cluster. They found out that determining whether clustering into K clusters satisfying a set of must-cut constraints is possible is equivalent to the graph K-coloring problem. Therefore, this task is NP-hard for $K \geq 3$. On the other hand, deciding whether a K-clustering exists that satisfies a set of must-join constraints can be solved efficiently in $\mathcal{O}(m+n)$ time where n is the number of clusters and m is the number of constraints. The complexity of other tasks arising from K-clustering with must-cut and must-join constraints is studied in their follow-up work (Davidson and Ravi, 2007).

Widely used metrics for comparison of two clusterings are for instance the Rand metric (Rand, 1971) and variation of information (Meilă, 2007). If we want to compare two multicuts of the same graph, we can compare them by applying Hamming metric (Hamming, 1950) on their characteristic functions. That is, we would count all edges where the two functions disagree whether the edge should belong to multicut or not. Our metrics presented in Section 3.5 can be viewed as a generalization of the Hamming metric and the Rand metric.

3.3 MULTICUTS AND LIFTED MULTICUTS

We start with providing a theoretical background for the multicut and lifted multicut problem that is more detailed than the brief introduction in Sections 2.1 and 2.2.

Definition 3.1. *Let $G = (V, E)$ be any graph. A subgraph $G' = (V', E')$ of G is called a* component *of G iff G' is non-empty, node-induced*[1]*, and connected*[2]*. A partition Π of V is called a* decomposition *of G iff for every $U \in \Pi$, the subgraph $(U, E \cap \binom{U}{2})$ of G induced by U is connected (and hence a component of G).*

[1]That is: $E' = E \cap \binom{V'}{2}$

[2]We do not require a component to be maximal w.r.t. the subgraph relation.

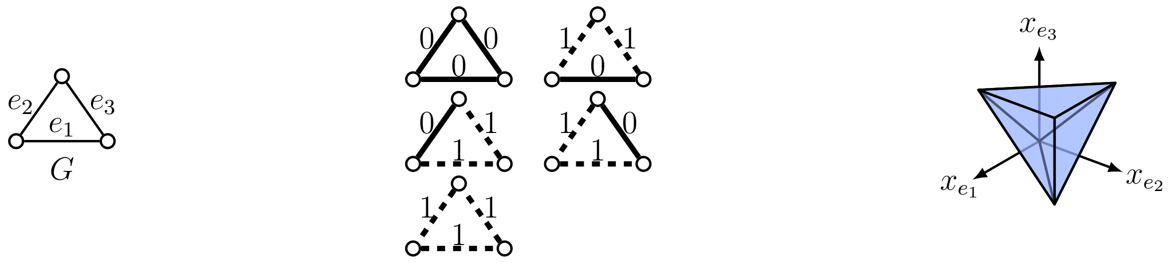


Figure 3.1: For any connected graph G (left), the characteristic functions of all multicuts of G (middle) span, as their convex hull in $\mathbb{R}^E$, the *multicut polytope* of G (right), a 0/1-polytope that is $|E|$-dimensional (Chopra and Rao, 1993).

For any graph G, we denote by $D_G \subset 2^{2^V}$ the set of all decompositions of G. Useful in the study of decompositions are the multicuts of a graph:

Definition 3.2. *For any graph $G = (V, E)$, a subset $M \subseteq E$ of edges is called a* multicut *of G iff, for every cycle $C \subseteq E$ of G, we have $|C \cap M| \neq 1$.*

Lemma 3.1. *(Chopra and Rao, 1993) It is sufficient in Definition 3.2 to consider only the* chordless *cycles.*

For any graph G, we denote by $M_G \subseteq 2^E$ the set of all multicuts of G. One reason why multicuts are useful in the study of decompositions is that, for every graph G, a one-to-one relation exists between the decompositions and the multicuts of G. An example is depicted in Figure 2.1.

Lemma 3.2. *For any graph $G = (V, E)$, the map $\phi_G : D_G \to 2^E$ defined by (3.1) is a bijection from D_G to M_G.*

$$\forall \Pi \in D_G \ \forall \{v, w\} \in E : \quad \{v, w\} \in \phi_G(\Pi) \ \Leftrightarrow \ \forall U \in \Pi (v \notin U \vee w \notin U) \tag{3.1}$$

Proof. First, we show that for any $\Pi \in D_G$, the image $\phi_G(\Pi)$ is a multicut of G. Assume the contrary, i.e. there exists a cycle C of G such that $|C \cap \phi_G(\Pi)| = 1$. Let $\{u, v\} = e \in C \cap \phi_G(\Pi)$, then for all $U \in \Pi$ it holds that $u \notin U$ or $v \notin U$. However, $C \setminus \{e\}$ is a sequence of edges $\{w_1, w_2\}, \dots, \{w_{k-1}, w_k\}$ such that $u = w_1, v = w_k$ and $\{w_i, w_{i+1}\} \notin \phi_G(\Pi)$ for all $1 \leq i \leq k-1$. Consequently, since Π is a partition of V, there exists some $U \in \Pi$ such that

$$w_1 \in U \wedge w_2 \in U \wedge \ldots \wedge w_{k-1} \in U \wedge w_k \in U.$$

This contradicts $w_1 = u \notin U$ or $w_k = v \notin U$.

To show injectivity of ϕ_G, let $\Pi = \{U_1, \dots, U_k\}$, $\Pi' = \{U'_1, \dots, U'_\ell\}$ be two decompositions of G. Suppose $\Pi \neq \Pi'$. Then (w.l.o.g.) there exist some $u, v \in V$ with $\{u, v\} \in E$ and some $U_i \in \Pi$ such that $u, v \in U_i$ and for all $U'_j \in \Pi'$ it holds that $u \notin U'_j$ or $v \notin U'_j$. Thus, $\{u, v\} \in \phi_G(\Pi')$ but $\{u, v\} \notin \phi_G(\Pi)$, which means $\phi_G(\Pi) \neq \phi_G(\Pi')$.

For surjectivity, take some multicut $M \subseteq E$ of G. Let $\Pi = \{U_1, \dots, U_k\}$ collect the node sets of the connected components of the graph $(V, E \setminus M)$. Apparently, Π defines a decomposition of G. We have $\{u, v\} \in \phi_G(\Pi)$ if and only if for all $U \in \Pi$ it holds that $v \notin U$ or $u \notin U$. The latter holds true if and only if $\{u, v\}$ is not contained in any connected component of $(V, E \setminus M)$, which is equivalent to $\{v, w\} \in M$. Hence, $\phi_G(\Pi) = M$.

□

Another reason why multicuts are useful in the study of decompositions is that for any graph $G = (V, E)$ and any decomposition Π of G, the characteristic function of the multicut induced by Π is a 0/1 encoding of Π of the fixed length $|E|$. Such encoding $x \in \{0, 1\}^E$ has to satisfy the cycle constraints (2.1). We denote the set of characteristic functions of multicuts of G as $X_G = \{x \in \{0, 1\}^E | x \text{ satisfies } (2.1)\}$.

3.3.1 COMPLETE GRAPHS

The decompositions of a complete graph $K_V := (V, \binom{V}{2})$ are precisely the partitions of the node set V (by Definition 3.1). The multicuts of a complete graph K_V relate one-to-one to the equivalence relations on V:

Lemma 3.3. *For any set V and the complete graph K_V, the map $\psi : M_{K_V} \to 2^{V \times V}$ defined by (3.2) is a bijection between M_{K_V} and the set of all equivalence relations on V.*

$$\forall M \in M_{K_V} \ \forall v, w \in V : \quad (v, w) \in \psi(M) \ \Leftrightarrow \ \{v, w\} \notin M \tag{3.2}$$

Proof. First, we show that for any $M \in M_{K_V}$ the image $\psi(M)$ is an equivalence relation on V. Since K_V is simple, we trivially have $\{v, v\} \notin M$ for any $v \in V$. Therefore, $(v, v) \in \psi(M)$, which means $\psi(M)$ is reflexive. Symmetry of $\psi(M)$ follows from $\{u, v\} = \{v, u\}$ for all $u, v \in V$. Now, suppose $(u, v), (v, w) \in \psi(M)$. Then $\{u, v, \}, \{v, w\} \notin M$ and thus $\{u, w\} \notin M$ (otherwise $C = \{u, v, w\}$ would be a cycle contradicting the definition of a multicut). Hence, $(u, w) \in \psi(M)$, which gives transitivity of $\psi(M)$.

Let M, M' be two multicuts of K_V with $\psi(M) = \psi(M')$. Then

$$\begin{aligned} \{u, v\} \in M &\iff (u, v) \notin \psi(M) \\ &\iff (u, v) \notin \psi(M') \\ &\iff \{u, v\} \in M'. \end{aligned}$$

Hence $M = M'$, so ψ is injective.

Let R be an equivalence relation on V and define M by

$$\{u, v\} \in M \iff (u, v) \notin R.$$

Transitivity of R implies that M is a multicut of K_V. Moreover, by definition, it holds that $\psi(M) = R$. Hence, ψ is also surjective.

□

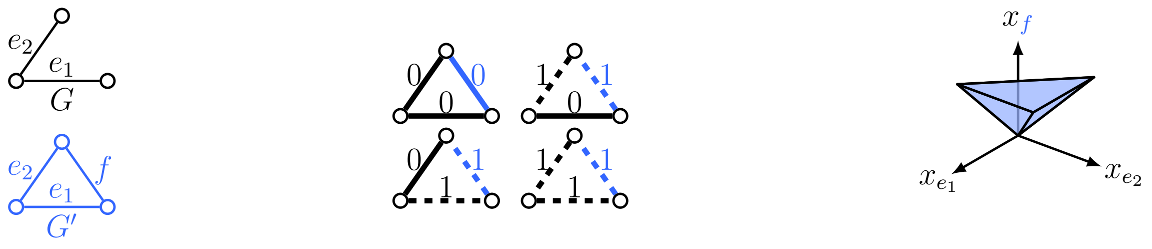


Figure 3.2: For any connected graph $G = (V, E)$ (top left) and any graph $G' = (V, E')$ with $E \subseteq E'$ (bottom left), those multicuts of G' (blue) that are lifted from G (middle) span, as their convex hull in $\mathbb{R}^E$, the *lifted multicut polytope* w.r.t. G and G' (right), a 0/1-polytope that is $|E'|$-dimensional (Theorem. 3.7).

The bijection between the decompositions of a graph and the multicuts of a graph (Lemma 3.2) specializes, for complete graphs, to the well-known bijection between the partitions of a set and the equivalence relations on the set (by Lemma 3.3). In this sense, decompositions and multicuts of graphs generalize partitions of sets and equivalence relations on sets.

3.3.2 EXTENSION TO LIFTED MULTICUTS

Definition 3.3. *For any graphs $G = (V, E)$ and $G' = (V, E')$ with $E \subseteq E'$, the composed map $\lambda_{GG'} := \phi_{G'} \circ \phi_G^{-1}$ is called the* lifting *of multicuts from G to G'.*

For any graphs $G = (V, E)$ and $G' = (V, E')$ with $E \subseteq E'$, we introduce the notation $F_{GG'} := E' \setminus E$, for brevity.

Lemma 3.4. *For any connected graph $G = (V, E)$, any graph $G' = (V, E')$ with $E \subseteq E'$ and any $x \in \{0, 1\}^{E'}$, the set $x^{-1}(1)$ is a multicut of G' lifted from G iff*

$$\forall C \in \text{cycles}(G)\, \forall e \in C :\; x_e \leq \sum_{e' \in C \setminus \{e\}} x_{e'} \tag{3.3}$$

$$\forall vw \in F_{GG'}\, \forall P \in vw\text{-paths}(G) :\; x_{vw} \leq \sum_{e \in P} x_e \tag{3.4}$$

$$\forall vw \in F_{GG'}\, \forall C \in vw\text{-cuts}(G) :\; 1 - x_{vw} \leq \sum_{e \in C} (1 - x_e) \tag{3.5}$$

Proof. Let $x \in \{0, 1\}^{E'}$ be such that $M' = x^{-1}(1)$ is a multicut of G' lifted from G. Every cycle in G is a cycle in G'. Moreover, for any path $vw = f \in F_{GG'}$ and any vw-path P in G, it holds that $P \cup \{f\}$ is a cycle in G'. Therefore, x satisfies all inequalities (3.3) and (3.4). Assume x violates some inequality of (3.5). Then there is an edge $vw \in F_{GG'}$ and some vw-cut C in G such that $x_{vw} = 0$ and for all $e \in C$ we have $x_e = 1$. Let Π be the partition of V corresponding to M' according to Lemma 3.2. There exists some $U \in \Pi$ with $v \in U$ and $w \in U$. However, for any $uu' = e \in C$ it holds that $u \notin U$ or $u' \notin U$. This means the subgraph $(U, E \cap \binom{U}{2})$ is not connected, as C is a vw-cut. Hence, Π is not a decomposition of G, which is a contradiction, because G is connected.

Now, suppose $x \in E'$ satisfies all inequalities (3.3)–(3.5). We show first that $M' = x^{-1}(1)$ is a multicut of G'. Assume the contrary, then there is a cycle C' in G' and some edge e' such that $C' \cap M' = \{e'\}$. For every $vw = f \in F_{GG'} \cap C' \setminus \{e'\}$ there exists a vw-path P in G such that $x_e = 0$ for all $e \in P$. Otherwise there would be some vw-cut in G violating (3.5), as G is connected. If we replace every such f with its associated path P in G, then the resulting cycle violates either (3.3) (if $e' \in E$) or (3.4) (if $e' \in F_{GG'}$). Thus, M' is a multicut of G'. By connectivity of G, the partition $\phi_{G'}^{-1}(M')$ is a decomposition of both G' and G. Therefore, $M = \lambda_{GG'}^{-1}(M') = \phi_G(\phi_{G'}^{-1}(M'))$ is a multicut of G and hence $M' = x^{-1}(1)$ is indeed lifted from G.

□

For any graphs $G = (V, E)$ and $G' = (V, E')$ with $E \subseteq E'$ we denote by $X_{GG'}$ the set of all $x \in \{0,1\}^{E'}$ that satisfy (3.3)–(3.5).

3.4 PARTIAL LIFTED MULTICUTS

As the first application of lifted multicuts, we study the classes of decompositions of a graph definable by must-join and must-cut constraints. For this, we consider partial functions. For any set E, a partial characteristic function of subsets of E is a function from any subset $F \subseteq E$ to $\{0,1\}$. With some abuse of notation, we denote the set of all partial characteristic functions of subsets of E by $\{0,1,*\}^E := \bigcup_{F \subseteq E} \{0,1\}^F$. For any $\tilde{x} \in \{0,1,*\}^E$, we denote the domain of $\tilde{x}$ by $\operatorname{dom} \tilde{x} := \tilde{x}^{-1}(\{0,1\})$.

For any connected graph $G = (V, E)$ whose decompositions we care about and any graph $G' = (V, E')$ with $E \subseteq E'$, we consider a partial function $\tilde{x} \in \{0,1,*\}^{E'}$. For any $\{v, w\} \in \operatorname{dom} \tilde{x}$, we constrain the nodes v and w to the same component if $\tilde{x}_{vw} = 0$ and to distinct components if $\tilde{x}_{vw} = 1$.

3.4.1 CONSISTENCY

A natural question to ask is whether a decomposition of the graph G exists that satisfies these constraints. We show that this decision problem is NP-complete.

Definition 3.4. *For any connected graph $G = (V, E)$, any graph $G' = (V, E')$ with $E \subseteq E'$, and any $\tilde{x} \in \{0,1,*\}^{E'}$, the elements of*

$$X_{GG'}[\tilde{x}] := \{x \in X_{GG'} \mid \forall e \in \operatorname{dom} \tilde{x} : \; x_e = \tilde{x}_e\} \tag{3.6}$$

are called the completions *of $\tilde{x}$ in $X_{GG'}$. In addition, $\tilde{x}$ is called* consistent *and a* partial characterization of multicuts of G' lifted from G *iff*

$$X_{GG'}[\tilde{x}] \neq \emptyset \, . \tag{3.7}$$

We denote the set of all partial characterizations of multicuts of G' lifted from G by

$$\tilde{X}_{GG'} := \left\{ \tilde{x} \in \{0,1,*\}^{E'} \;\middle|\; X_{GG'}[\tilde{x}] \neq \emptyset \right\} . \tag{3.8}$$

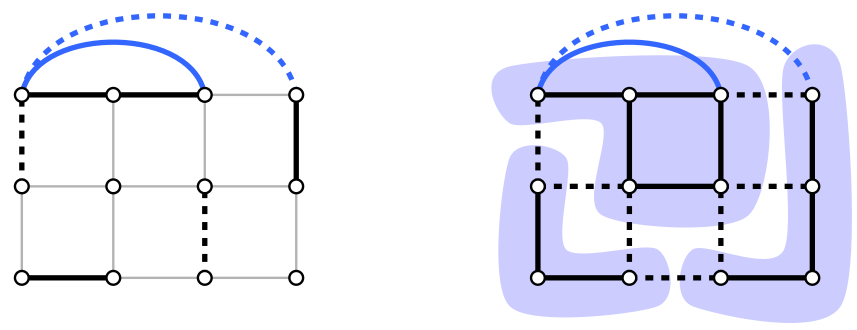


Figure 3.3: An example of a partial characterization of lifted multicut (left) and its possible completion (right), see Definition 3.4. Lifted edges are depicted in blue. Cut edges are thick and dashed, join edges are thick and solid. Unlabeled edges are thin and gray.

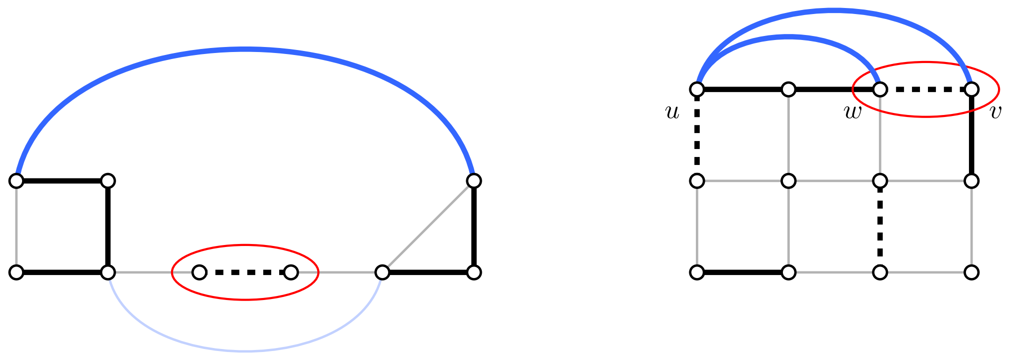


Figure 3.4: Examples of inconsistent must-join and must-cut constraints, see Definition 3.4. Lifted edges are depicted in blue. Cut edges are thick and dashed, join edges are thick and solid. Unlabeled edges are thin and gray resp. light blue if they are lifted. Red ellipses highlight the problematic edges.

Figure 3.3 shows an example of partial characterization of lifted multicut and its possible completion. Figure 3.4 depicts two examples where the must-join and must-cut constraints are not consistent.

Theorem 3.1. *Deciding consistency is NP-complete.*

Proof. Firstly, we show that the consistency problem is in NP. For that, we show that verifying, for any given $x \in \{0,1\}^{E'}$, that x is a completion of $\tilde{x}$ and a characteristic function of a multicut of G' lifted from G is a problem of polynomial time complexity. To verify that x is a completion of $\tilde{x}$, we verify for every $e \in \operatorname{dom} \tilde{x}$ that $x_e = \tilde{x}_e$. This takes time $O(|E|)$. To verify that $x^{-1}(1)$ is a multicut of G' lifted from G, we employ a disjoint set data structure initialized with singleton sets V. For any $\{v,w\} \in x^{-1}(0)$, we call $\operatorname{union}(v,w)$. Then, we verify for every $\{v,w\} \in x^{-1}(1)$ that $\operatorname{find}(v) \neq \operatorname{find}(w)$. This takes time $O(|E| + |V| \log |V|)$.

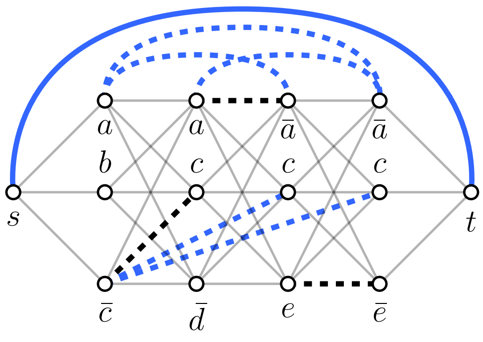


Figure 3.5: To show that the consistency problem is NP-complete (Theorem 3.1), we reduce 3-SAT to this problem. Shown above is the instance of the consistency problem constructed for the instance of 3-SAT given by the form $(a \vee b \vee \bar{c}) \wedge (a \vee c \vee \bar{d}) \wedge (\bar{a} \vee c \vee e) \wedge (\bar{a} \vee c \vee \bar{e})$. Lifted edges are depicted in blue. Cut edges ($\tilde{x}_e = 1$) are thick and dashed, join edges ($\tilde{x}_e = 0$) are thick and solid. Unlabeled edges ($e \notin \operatorname{dom} \tilde{x}$) are thin and gray.

To show that the consistency problem is NP-hard, we reduce 3-SAT to this problem. For that, we consider any instance of 3-SAT defined by a propositional logic formula A in 3-SAT form. An example is shown in Figure 3.5. Let m be the number of variables and n the number of clauses in A.

In order to define an instance of the consistency problem w.r.t. this instance of 3-SAT, we construct in polynomial time a connected graph $G = (V, E)$, a graph $G' = (V, E')$ with $E \subseteq E'$, and a partial characteristic function $\tilde{x} \in \{0, 1, *\}^{E'}$ as described below. An example of this construction is shown in Figure 3.5.

- There are $3n + 2$ nodes in V. Two nodes are denoted by s and t. Additional nodes are organized in n layers. For $j \in \{1, \dots, n\}$, the j-th layer corresponds to the j-th clause in A, containing one node for each of the three literals[3] in the clause. Every node is labeled with its corresponding literal. Layer 0 contains only the node s. Layer $n + 1$ contains only the node t.

- Any two consecutive layers are connected such that their nodes together induce a complete bipartite subgraph of G. Additionally, any nodes v and w labeled with conflicting literals (e.g. a and $\bar{a}$) that are not already connected in G are connected in G' by an edge $\{v, w\} \in E' \setminus E$.

- For any edge $\{v, w\} \in E'$ whose nodes v and w are labeled with conflicting literals, we set $\tilde{x}_{vw} = 1$. In addition, we introduce the edge $\{s, t\} \in E' \setminus E$ and define $\tilde{x}_{st} = 0$. No other edges are in the domain of $\tilde{x}$.

Observe that $\tilde{x}$ is consistent iff there exists an st-path P in G such that no edge or chord $\{v, w\}$ of P is such that $\tilde{x}_{vw} = 1$. Any such path is called feasible. All other st-paths in G are called infeasible.

Now, we show firstly that the existence of a feasible path implies the existence of a solution to the given instance of 3-SAT. Secondly, we show that the existence of

[3]A literal is either a variable a or a negated variable $\bar{a}$.

a solution to the given instance of 3-SAT implies the existence of a feasible path. That suffices.

1. Let P be a feasible path and let V_P be its node set. An assignment χ to the variables of the instance of 3-SAT is constructed as follows: For any node $v \in V_P$ whose label is a variable a, we define $\chi(a) :=$ true. For any node $v \in V_P$ whose label is a negated variable $\bar{a}$, we define $\chi(a) :=$ false. All remaining variables are assigned arbitrary truth values. By the properties of P, χ is well-defined and $A[\chi]$ is true.

2. Let χ be a solution to the given instance of 3-SAT. As every clause of A contains one literal that is true, and by the construction of G, we can choose an st-path in G along which all nodes are labeled with literals that are true for the assignment χ. By virtue of χ being a solution to the instance of 3-SAT, any pair of literals that are both true are non-conflicting. Thus, P has no edge or chord $\{v, w\}$ such that $\tilde{x}_{vw} = 1$.

□

Lemma 3.5. *Consistency can be decided efficiently if $E \subseteq \operatorname{dom} \tilde{x}$ or*

$$\forall vw \in \operatorname{dom} \tilde{x} \setminus E : \quad \tilde{x}_{vw} = 1 \vee \exists P \in vw\text{-path}(G)\, \forall e \in P : \, \tilde{x}_e = 0 \tag{3.9}$$

Proof. Firstly, suppose that $E \subseteq \operatorname{dom} \tilde{x}$. In this case, it is clear that $\tilde{x}$ is consistent iff $\tilde{x}$ satisfies all cycle inequalities (3.3) w.r.t. the graph $(V, E \cap \operatorname{dom} \tilde{x})$. This can be checked in time $O(|V| + |E'|)$ as follows: Label the maximal components of the subgraph $G_{\tilde{x}}$ of G induced by the edge set $\{e \in E : \tilde{x}_e = 0\}$. Then, for every $\{v, w\} \in E'$ with $\tilde{x}_{vw} = 1$, check if v and w are in distinct maximal components of $G_{\tilde{x}}$. If so, $\tilde{x}$ is consistent, otherwise, $\tilde{x}$ is inconsistent.

Now, suppose $\tilde{x} \in \{0, 1, *\}^{E'}$ satisfies (3.9). We show that, similar to the first case, $\tilde{x}$ is consistent iff all inequalities (3.3) and (3.4) are satisfied w.r.t. the graph $(V, E' \cap \operatorname{dom} \tilde{x})$. This can be checked analogously to the first case.

The necessity of this condition is clear. To show sufficiency, assume this condition holds true. We construct some $x \in X_{GG'}[\tilde{x}]$ as follows. For all $e \in \operatorname{dom} \tilde{x}$, set $x_e := \tilde{x}_e$. For all $\{v, w\} = f \in E' \setminus E$ such that $f \notin \operatorname{dom} \tilde{x}$ and such that there is a vw-path P in G with $\tilde{x}_e = 0$ for all $e \in P$, set $x_f := 0$. For all remaining edges e, set $x_e := 1$. By construction, x satisfies (3.3), (3.4) and (3.5).

□

3.4.2 SPECIFICITY

A less obvious question to ask for any partial characterization $\tilde{x}$ of multicuts of G' lifted from G is whether $\tilde{x}$ is maximally specific for its completions in $X_{GG'}$. In other words, is there no edge $e \in E' \setminus \operatorname{dom} \tilde{x}$ such that, for any completions x, x' of $\tilde{x}$ in $X_{GG'}$, we have $x_e = x'_e$, i.e., an edge that could be included in $\operatorname{dom} \tilde{x}$ without changing the set of completions of $\tilde{x}$ in $X_{GG'}$? We show that deciding maximal specificity is NP-hard.

Definition 3.5. *Let $G = (V, E)$ a connected graph and $G' = (V, E')$ a graph with $E \subseteq E'$. For any $\tilde{x} \in \tilde{X}_{GG'}$, the edges*

$$E'[\tilde{x}] := \{e \in E' \,|\, \forall x, x' \in X_{GG'}[\tilde{x}] : \, x_e = x'_e\} \tag{3.10}$$

are called decided. *The edges* $E' \setminus E'[\tilde{x}]$ *are called* undecided. *Moreover,* $\tilde{x}$ *is called* maximally specific *iff*[4]

$$E'[\tilde{x}] \subseteq \operatorname{dom} \tilde{x}\,. \tag{3.11}$$

Theorem 3.2. *Deciding maximal specificity is NP-hard.*

Proof. To show that the maximal specificity problem is NP-hard, we reduce 3-SAT to this problem: For any given instance of 3-SAT we construct in polynomial time a connected graph $G = (V, E)$, a graph $G' = (V, E')$ with $E \subseteq E'$, and a partial characteristic function $\tilde{x} \in \{0, 1, *\}^{E'}$ as in the proof of Theorem 3.1, except that now, we let $st \notin \operatorname{dom} \tilde{x}$.

We know that $\tilde{x}$ is consistent because the vector of ones $\mathbb{1} \in X_{GG'}[\tilde{x}]$. We show that $\tilde{x}$ is maximally specific iff the given instance of 3-SAT has a solution:

Firstly, every $e \in E' \setminus (\operatorname{dom} \tilde{x} \cup \{st\})$ is undecided, by the following argument: (i) There exists an $x \in X_{GG'}[\tilde{x}]$ with $x_e = 1$, namely $\mathbb{1}$. (ii) There exists an $x \in X_{GG'}[\tilde{x}]$ with $x_e = 0$, namely the $x \in \{0, 1\}^{E'}$ with $x_e = 0$ and $\forall f \in E' \setminus \{e\} : x_f = 1$. To see that $x \in X_{GG'}[\tilde{x}]$, observe that $e \in E$ and $\tilde{x}^{-1}(0) = \emptyset$. Thus, st is the only edge in $E' \setminus \operatorname{dom} \tilde{x}$ that is possibly decided. That is:

$$E'[\tilde{x}] \subseteq \{st\} \cup \operatorname{dom} \tilde{x} \tag{3.12}$$

Thus, $\tilde{x}$ is maximally specific iff $\tilde{x}$ is undecided. More specifically, $\tilde{x}$ is maximally specific iff there exists an $x \in X_{GG'}[\tilde{x}]$ with $x_{st} = 0$, as we know of the existence of $\mathbb{1} \in X_{GG'}[\tilde{x}]$. Thus, $\tilde{x}$ is maximally specific iff the given instance of 3-SAT has a solution, by the arguments made in the proof of Theorem 3.1.

□

Below, we justify the term *maximal specificity* and define an operation that maps any partial characterization of lifted multicuts to one that is maximally specific.

Definition 3.6. *For any connected graph* $G = (V, E)$ *and any graph* $G' = (V, E')$ *with* $E \subseteq E'$*, the relation* $\leq$ *on* $\tilde{X}_{GG'}$ *defined by* (3.13) *is called the* specificity *of partial characterizations of multicuts of* G' *lifted from* G*.*

$$\forall \tilde{x}, \tilde{x}' \in \tilde{X}_{GG'} : \quad \tilde{x} \leq \tilde{x}' \Leftrightarrow \operatorname{dom} \tilde{x} \subseteq \operatorname{dom} \tilde{x}' \wedge \forall e \in \operatorname{dom} \tilde{x} : \tilde{x}_e = \tilde{x}'_e \tag{3.13}$$

Lemma 3.6. *For any connected graph* $G = (V, E)$ *and any graph* $G' = (V, E')$ *with* $E \subseteq E'$*, specificity is a partial order on* $\tilde{X}_{GG'}$*.*

Proof. Reflexivity is obvious. Antisymmetry: $(\tilde{x} \leq \tilde{x}' \wedge \tilde{x}' \leq \tilde{x}) \Rightarrow (\operatorname{dom} \tilde{x} = \operatorname{dom} \tilde{x}' \wedge \forall e \in \operatorname{dom} \tilde{x} : \tilde{x}_e = \tilde{x}'_e)$. Transitivity: Let $\tilde{x} \leq \tilde{x}' \leq \tilde{x}''$. Then $\operatorname{dom} \tilde{x} \subseteq \operatorname{dom} \tilde{x}' \subseteq \operatorname{dom} \tilde{x}''$ and $\forall e \in \operatorname{dom} \tilde{x} : \tilde{x}_e = \tilde{x}'_e = \tilde{x}''_e$.

□

[4]Note that (3.11) is equivalent to $E'[\tilde{x}] = \operatorname{dom} \tilde{x}$, as $E'[\tilde{x}] \supseteq \operatorname{dom} \tilde{x}$ holds by definition of $E'[\tilde{x}]$.

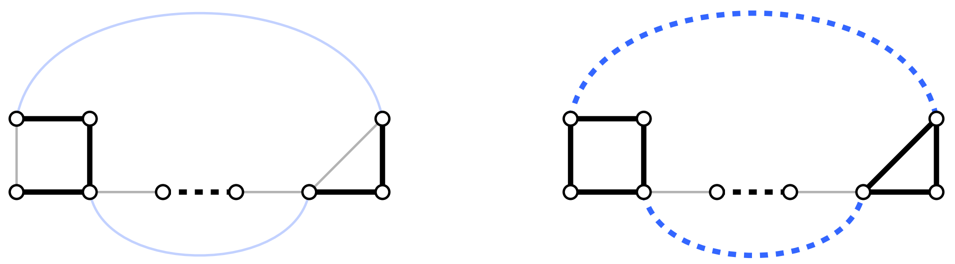


Figure 3.6: Example of a partial characterization of lifted multicut (left) and its closure (right), see Definition 3.7. Lifted edges are depicted in blue. Cut edges ($\tilde{x}_e = 1$) are thick and dashed, join edges ($\tilde{x}_e = 0$) are thick and solid. Unlabeled edges ($e \notin \operatorname{dom} \tilde{x}$) are thin and gray resp. light blue if they are lifted.

Note that two partial characterizations $\tilde{x}, \tilde{x}' \in \tilde{X}_{GG'}$ with the same completions $X_{GG'}[\tilde{x}] = X_{GG'}[\tilde{x}']$ need not be comparable w.r.t. $\leq$. For example, consider the graphs G, G' from Figure 3.2, consider $\tilde{x} : e_1 \mapsto 0, e_2 \mapsto 0$ and $\tilde{x}' : f \mapsto 0$. Nevertheless, we have the following lemma.

Lemma 3.7. *For any connected graph $G = (V, E)$, any graph $G' = (V, E')$ with $E \subseteq E'$, any $\tilde{x} \in \tilde{X}_{GG'}$ and*

$$\tilde{X}_{GG'}[\tilde{x}] := \left\{ \tilde{x}' \in \tilde{X}_{GG'} \,\middle|\, X_{GG'}[\tilde{x}'] = X_{GG'}[\tilde{x}] \right\} \tag{3.14}$$

a maximum of $\tilde{X}_{GG'}[\tilde{x}]$ w.r.t. $\leq$ exists and is unique. Moreover, $\tilde{x}$ is maximally specific in the sense of Definition 3.5 iff $\tilde{x}$ is maximal w.r.t. $\leq$ in $\tilde{X}_{GG'}[\tilde{x}]$.

Proof. We show first that $\tilde{x}'$ is maximal w.r.t. $\leq$ in $\tilde{X}_{GG'}[\tilde{x}]$ iff it is maximally specific. This implies the existence and uniqueness of the maximum of $\tilde{X}_{GG'}[\tilde{x}]$ by construction via $\operatorname{dom} \tilde{x}' = E'[\tilde{x}]$.

Let $\tilde{x}' \in \tilde{X}_{GG'}[\tilde{x}]$ be maximally specific and suppose $\tilde{x}' \leq \tilde{x}''$ for some $\tilde{x}'' \in \tilde{X}_{GG'}[\tilde{x}]$. Then $\operatorname{dom} \tilde{x}'' = \operatorname{dom} \tilde{x}'$, since $X_{GG'}[\tilde{x}'] \neq X_{GG'}[\tilde{x}'']$ if $\operatorname{dom} \tilde{x}'' \setminus E'[\tilde{x}] \neq \emptyset$. Thus, $\tilde{x}' = \tilde{x}''$, which means $\tilde{x}'$ is maximal w.r.t. $\leq$ in $\tilde{X}_{GG'}[\tilde{x}]$.

Conversely, any maximal element $\tilde{x}'$ of $\tilde{X}_{GG'}[\tilde{x}]$ w.r.t. $\leq$ must satisfy $E'[\tilde{x}] \subseteq \operatorname{dom} \tilde{x}'$, which means it is maximally specific.

Hence, the unique maximum $\tilde{x}' \in \tilde{X}_{GG'}[\tilde{x}]$ is obtained as follows. For an arbitrary $x \in X_{GG'}[\tilde{x}]$ define $\tilde{x}'$ via $\tilde{x}'_e := x_e$ for all decided edges $e \in E'[\tilde{x}]$.

□

Definition 3.7. *Let $G = (V, E)$ be a connected graph and let $G' = (V, E')$ be a graph with $E \subseteq E'$. For any $\tilde{x} \in \tilde{X}_{GG'}$, we call the unique maximum of $\tilde{X}_{GG'}[\tilde{x}]$ w.r.t. $\leq$ the* closure *of $\tilde{x}$ w.r.t. G and G' and denote it by $\operatorname{cl}_{GG'} \tilde{x}$.*

We denote by $\hat{X}_{GG'}$ the set of all maximally specific partial characterizations of multicuts of G' lifted from G, i.e.:

$$\hat{X}_{GG'} := \left\{ \tilde{x} \in \tilde{X}_{GG'} \,\middle|\, \tilde{x} = \operatorname{cl}_{GG'} \tilde{x} \right\} . \tag{3.15}$$

Figure 3.6 shows an example of a partial characterization of lifted multicut together with its closure.

Theorem 3.3. *For any $\tilde{x}, \tilde{x}' \in \tilde{X}_{GG'}$, we have $X_{GG'}[\tilde{x}] = X_{GG'}[\tilde{x}'] \Leftrightarrow \tilde{X}_{GG'}[\tilde{x}] = \tilde{X}_{GG'}[\tilde{x}'] \Leftrightarrow \operatorname{cl}_{GG'} \tilde{x} = \operatorname{cl}_{GG'} \tilde{x}'$.*

Proof. Let us have $\tilde{x}, \tilde{x}' \in \tilde{X}_{GG'}$.

- The implication $X_{GG'}[\tilde{x}] = X_{GG'}[\tilde{x}'] \Rightarrow \tilde{X}_{GG'}[\tilde{x}] = \tilde{X}_{GG'}[\tilde{x}']$: follows from the definition of $\tilde{X}_{GG'}[\tilde{x}]$ in Lemma 3.7.
- The implication $\tilde{X}_{GG'}[\tilde{x}] = \tilde{X}_{GG'}[\tilde{x}'] \Rightarrow \operatorname{cl}_{GG'} \tilde{x} = \operatorname{cl}_{GG'} \tilde{x}'$ follows from the definition of the closure of $\tilde{x}$ as the maximum of $\tilde{X}_{GG'}[\tilde{x}]$.
- The implication $\operatorname{cl}_{GG'} \tilde{x} = \operatorname{cl}_{GG'} \tilde{x}' \Rightarrow X_{GG'}[\tilde{x}] = X_{GG'}[\tilde{x}']$ follows from $\operatorname{cl}_{GG'} \tilde{x} = \operatorname{cl}_{GG'} \tilde{x}' \in \tilde{X}_{GG'}[\tilde{x}]$.

□

Lemma 3.8. *For any connected graph $G = (V, E)$, any graph $G' = (V, E')$ with $E \subseteq E'$ and any $x \in X_G$, the closure $y := \operatorname{cl}_{GG'} x$ of x w.r.t. G and G' coincides with the lifting of the multicut $x^{-1}(1)$ of G to the multicut $y^{-1}(1)$ of G', i.e.*

$$(\operatorname{cl}_{GG'} x)^{-1}(1) = \lambda_{GG'}(x^{-1}(1)) \,. \tag{3.16}$$

Proof. Let $x \in X_G$ and define $y = \operatorname{cl}_{GG'} x$. Since $\operatorname{dom} x = E$, it holds that $E'[x] = E'$, i.e. all edges are decided. Therefore, $y^{-1}(1)$ is a multicut of G' and for all $\{v, w\} = f \in E' \setminus E$ it holds that $y_f = 0$ iff there is a vw-path P in G such that $x_e = 0$ for all $e \in P$. By Lemma 3.4, this implies $y^{-1}(1) = \lambda_{GG'}(x^{-1}(1))$.

□

Theorem 3.4. *Computing closures is NP-hard.*

Proof. Computing closures is at least as hard as deciding maximal specificity: To decide maximal specificity of $\tilde{x} \in \tilde{X}_{GG'}$, compute its closure $\operatorname{cl}_{GG'} \tilde{x}$. Then $\tilde{x}$ is maximally specific iff $\operatorname{dom} \tilde{x} = \operatorname{dom} \operatorname{cl}_{GG'} \tilde{x}$, i.e., if $\tilde{x} = \operatorname{cl}_{GG'} \tilde{x}$. By Theorem 3.2, this means computing closures is NP-hard.

□

Lemma 3.9. *In the special case that $E' = E$ or $E \subseteq \operatorname{dom} \tilde{x}$, the closure can be computed efficiently.*

Proof. Let $\tilde{x} \in \tilde{X}_{GG'}$ and $\tilde{y} = \operatorname{cl}_{GG'} \tilde{x}$.

Suppose first that $E = E'$. We describe how to compute $\tilde{y}$ efficiently. Obviously, we must set $\tilde{y}_e = \tilde{x}_e$ for all $e \in \operatorname{dom} \tilde{x}$. Furthermore, we must set $\tilde{y}_{vw} = 0$ for all $\{v, w\} \in E \setminus \operatorname{dom} \tilde{x}$ such that there is a vw-path P in G with $\tilde{x}_e = 0$ for all $e \in P$. Moreover, we must set $\tilde{y}_{vw} = 1$ for all $\{v, w\} \in E \setminus \operatorname{dom} \tilde{x}$ that satisfy

$$\exists P \in vw\text{-paths}(G)\ \exists e \in P : \qquad \tilde{x}_e = 1 \ \wedge\ \forall e' \in P \setminus \{e\} : \tilde{x}_{e'} = 0 \,. \tag{3.17}$$

Therefore, initialize a disjoint-set data structure with singleton sets V. Apply the union operation on all edges $e \in \operatorname{dom} \tilde{x}$ where $\tilde{x}_e = 0$, i.e. *contract* all 0-labeled edges. Then, set $\tilde{y}_e = 0$ for all edges that connect nodes of the same component. If there is an edge e' between two components such that $\tilde{x}_{e'} = 1$, then for all edges e between those components set $\tilde{y}_e = 1$. The remaining edges are undecided by $\tilde{x}$. In case we only want to decide maximal specificity, we can stop upon finding the first edge $e \in \operatorname{dom} \tilde{y} \setminus \operatorname{dom} \tilde{x}$.

Now suppose that $E \subseteq \operatorname{dom} \tilde{x}$. In this case, all edges are decided, because $\tilde{x}|_E \in X_G$. According to Lemma 3.8, the closure $\tilde{y}$ corresponds to the lifting of $\tilde{x}|_E$ to G'. Therefore, to obtain $\tilde{y}$, compute the decomposition of G associated to $\tilde{x}|_E$ using, e.g., a disjoint-set data structure. Set $\tilde{y}_e = 0$ if e is an edge within a component. Set $\tilde{y}_e = 1$ if e is an edge between components.

□

Lemma 3.10. *Maximal specificity can be decided efficiently if $E' = E$ or $E \subseteq \operatorname{dom} \tilde{x}$.*

Proof. Observe that $\tilde{x}$ is maximally specific iff $\operatorname{cl}_{GG'} \tilde{x} = \tilde{x}$. Thus, Lemma 3.10 follows from Lemma 3.9. □

3.5 METRICS

3.5.1 METRICS ON DECOMPOSITIONS

As the second application of lifted multicuts, we compare decompositions of a given graph by comparing lifted multicuts that characterize these decompositions. We compare these lifted multicuts by comparing their characteristic functions by Hamming metrics: For any $E' \neq \emptyset$ and any $e \in E'$, we define $d^1_e, d^1_{E'} : \{0,1\}^{E'} \times \{0,1\}^{E'} \to \mathbb{N}^+_0$ by the forms

$$d^1_e(x, x') = \begin{cases} 0 & \text{if } x_e = x'_e \\ 1 & \text{otherwise} \end{cases} \tag{3.18}$$

$$d^1_{E'}(x, x') = \sum_{e \in E'} d^1_e(x, x') \,. \tag{3.19}$$

Theorem 3.5. *For any connected graph $G = (V, E)$, any graph $G' = (V, E')$, any $\mu : E' \to \mathbb{R}^+$, the set $E'' := E \cup E'$ and the graph $G'' := (V, E'')$, the function $d^\mu_{E'} : X_{GG''} \times X_{GG''} \to \mathbb{R}^+_0$ of the form (3.20) is a pseudo-metric on $X_{GG''}$. $d^\mu_{E'}$ is a metric on $X_{GG''}$ iff G' is a supergraph of G (i.e. iff $E \subseteq E'$).*

$$d^\mu_{E'}(x, x') := \sum_{e \in E'} \mu_e \, d^1_e(x, x') \tag{3.20}$$

Proof. Symmetry and non-negativity follow directly from the definition, and so does $d^\mu_{E'}(x, x) = 0$ for all $x \in X_{GG''}$. For any $e \in E'$, the form d^1_e on $E' \times E'$ is a Hamming metric on words of length 1 from the alphabet $\{0, 1\}$. Therefore, it satisfies the triangle inequality. Hence, for any $x, y, z \in X_{GG''}$:

$$\begin{aligned} d^{\mu}_{E'}(x,z) &= \sum_{e \in E'} \mu_e d^1_e(x,z) \leq \sum_{e \in E'} \mu_e (d^1_e(x,y) + d^1_e(y,z)) = \\ &= \sum_{e \in E'} \mu_e d^1_e(x,y) + \sum_{e \in E'} \mu_e d^1_e(y,z) = d^{\mu}_{E'}(x,y) + d^{\mu}_{E'}(y,z)\,. \end{aligned} \tag{3.21}$$

Thus, $d^{\mu}_{E'}$ is a pseudo-metric on $X_{GG''}$.

If $E \subseteq E'$, then $G' = G''$ and thus, $X_{GG''} = X_{GG'} \subseteq X_{G'}$. For any two $x, x' \in X_{GG''} \subseteq X_{G'}$, it holds that $d^{\mu}_{E'}(x,x') = 0$ iff $d^1_e(x,x') = 0$ for all $e \in E'$, i.e. iff $x = x'$. Conversely, suppose there exists some $e \in E \setminus E'$. Define $x, x' \in X_{GG''}$ via $x_{e'} = x'_{e'} = 1$ for all $e' \in E'' \setminus \{e\}$ and $x_e = 1$, $x'_e = 0$. It holds that $x \neq x'$ but $d^{\mu}_{E'}(x,x') = 0$.

□

By the one-to-one relation between decompositions and multicuts (Lemma 3.2), $d^{\mu}_{E'}$ induces a pseudo-metric on the set D_G of all decompositions of G. Two special cases are well-known: For $E' = E$ and $\mu = 1$, we have $d^{\mu}_{E'} = d^1_E$, which is the Hamming metric (3.19) on the multicuts that characterize the decompositions, also known as the boundary metric on decompositions. For $E' = \binom{V}{2}$ and $\mu = 1$, $d^1_{E'}$ specializes to the metric of Rand (1971). Between these extremes, i.e., for $E \subseteq E' \subseteq \binom{V}{2}$, the metric $d^{\mu}_{E'}$ can be used to analyze more specifically how two decompositions of the same graph differ. We propose an analysis w.r.t. the distance δ_{vw} of nodes v and w in G, i.e., w.r.t. the length of the shortest vw-path in G. For this, we denote by $\delta_G := \max\{\delta_{vw} : vw \in \binom{V}{2}\}$ the diameter of G.

Definition 3.8. *For any connected graph $G = (V,E)$ and any $n \in \mathbb{N}$, let $E[n] := \{vw \in \binom{V}{2} \mid \delta_{vw} = n\}$ the set of pairs of nodes of distance n in G. Moreover, let $\mu^n : E[n] \to \mathbb{Q}^+$ the constant function that maps any $vw \in E[n]$ to $1/|E[n]|$. For any connected graph $G = (V,E)$, we call the sequence*

$$\left(d^{\mu^n}_{E[n]}\right)_{n \in \{1,\dots,\delta_G\}} \tag{3.22}$$

the spectrum of pseudo-metrics *on decompositions of G. For $E' := \binom{V}{2}$ and $\mu : E' \to \mathbb{Q}^+ : vw \mapsto 1/(\delta_G|E[\delta_{vw}]|)$, we call the metric $d^{\mu}_{E'}$ the* δ-metric *on decompositions of G.*

An example of a spectrum of pseudo-metrics is depicted in Figure 3.7. For any two decompositions Π, Π' of a connected graph G and suitable lifted multicuts x, x' characterizing these decompositions, $d^{\mu^n}_{E[n]}(x,x')$ equals the fraction of pairs of nodes at distance n in G that are either cut by Π and joined by Π', or cut by Π' and joined by Π. That is, the pseudo-metric $d^{\mu^n}_{E[n]}$ compares decompositions of G specifically w.r.t. the distance n in G. The δ-metric compares decompositions w.r.t. all distances, and each distance is weighted equally. This is in contrast to the Rand metric which is also a comparison w.r.t. all distances but each distance is weighted by the number of pairs of nodes that have this distance.

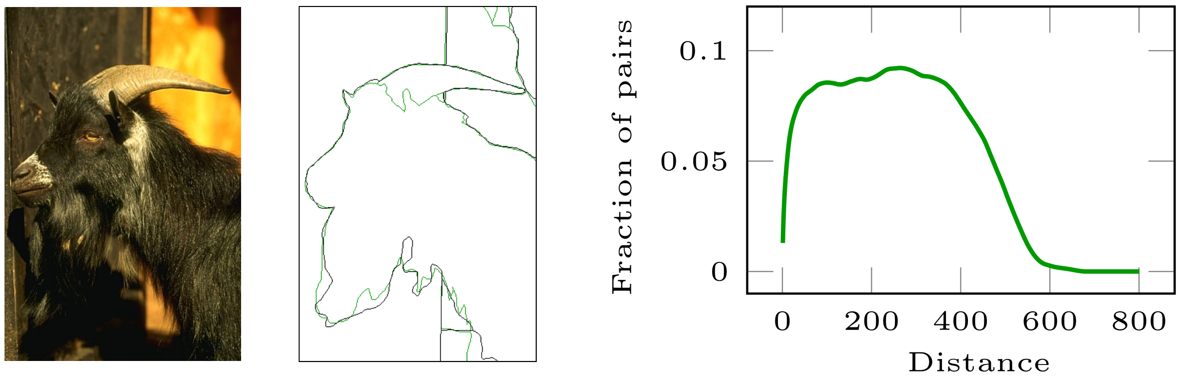


Figure 3.7: Depicted are two decompositions of the pixel grid graph of an image, all from Arbeláez *et al.* (2011), along with the spectrum of pseudo-metrics of these decompositions (Definition 3.8).

3.5.2 METRICS ON CLASSES OF DECOMPOSITIONS

We compare classes of decompositions definable by must-join and must-cut constraints by comparing partial lifted multicuts that characterize these decompositions. To compare partial lifted multicuts, we compare their partial characteristic functions by an extension of the Hamming metric: For any $E' \neq \emptyset$, any $e \in E'$ and any $\theta \in \mathbb{R}_0^+$, we define $d_e^\theta, d_{E'}^\theta : \{0,1,*\}^{E'} \times \{0,1,*\}^{E'} \to \mathbb{R}_0^+$ such that for all $\tilde{x}, \tilde{x}' \in \{0,1,*\}^{E'}$:

$$d_e^\theta(\tilde{x},\tilde{x}') = \begin{cases} 1 & \text{if } e \in \operatorname{dom}\tilde{x} \wedge e \in \operatorname{dom}\tilde{x}' \wedge \tilde{x}_e \neq \tilde{x}'_e \\ 0 & \text{if } e \in \operatorname{dom}\tilde{x} \wedge e \in \operatorname{dom}\tilde{x}' \wedge \tilde{x}_e = \tilde{x}'_e \\ 0 & \text{if } e \notin \operatorname{dom}\tilde{x} \wedge e \notin \operatorname{dom}\tilde{x}' \\ \theta & \text{otherwise} \end{cases} \tag{3.23}$$

$$d_{E'}^\theta(\tilde{x},\tilde{x}') = \sum_{e \in E'} d_e^\theta(\tilde{x},\tilde{x}')\,. \tag{3.24}$$

Theorem 3.6. *For any connected graph $G = (V,E)$, any graph $G' = (V,E')$ with $E \subseteq E'$ and any $\theta \in [\frac{1}{2},1]$, the function $\tilde{d}_{E'}^\theta : \tilde{X}_{GG'} \times \tilde{X}_{GG'} \to \mathbb{R}_0^+$ of the form*

$$\tilde{d}_{E'}^\theta(\tilde{x},\tilde{x}') := d_{E'}^\theta(\operatorname{cl}_{GG'}\tilde{x}, \operatorname{cl}_{GG'}\tilde{x}') \tag{3.25}$$

is a pseudo-metric on $\tilde{X}_{GG'}$ and a metric on $\hat{X}_{GG'}$. Moreover, for any $\tilde{x},\tilde{x}' \in \tilde{X}_{GG'}$:

$$\tilde{X}_{GG'}[\tilde{x}] = \tilde{X}_{GG'}[\tilde{x}'] \Leftrightarrow \tilde{d}_{E'}^\theta(\tilde{x},\tilde{x}') = 0\,. \tag{3.26}$$

Proof. We first prove that $\tilde{d}_{E'}^\theta$ is a metric on $\hat{X}_{GG'}$. For any $\tilde{x} \in \hat{X}_{GG'}$, we have $\operatorname{cl}_{GG'}\tilde{x} = \tilde{x}$. Thus, for all $\tilde{x},\tilde{x}' \in \hat{X}_{GG'}$, we have $\tilde{d}_{E'}^\theta(\tilde{x},\tilde{x}') = d_{E'}^\theta(\tilde{x},\tilde{x}')$. Therefore, positive definiteness and symmetry are obvious from the definition of $d_{E'}^\theta(\tilde{x},\tilde{x}')$. To establish the triangle inequality for $d_{E'}^\theta$, we prove it for θd_e^1 and any edge $e \in E'$. Let $\tilde{x},\tilde{y},\tilde{z} \in \hat{X}_{GG'}$ and consider the inequality

$$\theta d_e^1(\tilde{x},\tilde{z}) \leq \theta d_e^1(\tilde{x},\tilde{y}) + \theta d_e^1(\tilde{y},\tilde{z}). \tag{3.27}$$

$\tilde{x}_e$	$\tilde{y}_e$	$\tilde{z}_e$	lhs	rhs
0	0	0	0	0
1	1	1	0	0
*	*	*	0	0
0	*	1	1	2θ
0	1/0	1	1	1
0	0/*	*	θ	θ
1	1/*	*	θ	θ

Table 3.1: The left- and right-hand side of the inequality $\theta d_e^1(\tilde{x}, \tilde{z}) \leq \theta d_e^1(\tilde{x}, \tilde{y}) + \theta d_e^1(\tilde{y}, \tilde{z})$ for all possible combinations of values $\tilde{x}_e, \tilde{y}_e, \tilde{z}_e$ where $\tilde{x}, \tilde{y}, \tilde{z} \in \hat{X}_{GG'}$. The right-hand side is always greater or equal to the left-hand side iff $0.5 \leq \theta$.

In Table 3.1, the left-hand side and right-hand side of (3.27) are evaluated for all possible assignments of values to $\tilde{x}_e, \tilde{y}_e, \tilde{z}_e$. It is obvious from this table that (3.27) holds iff $\theta \geq 0.5$.

We now show that $\tilde{d}_{E'}^\theta$ is a pseudo-metric on $\tilde{X}_{GG'}$. Symmetry and non-negativity are obvious from the definition. For all $\tilde{x} \in \tilde{X}_{GG'}$, we have $\tilde{d}_{E'}^\theta(\tilde{x}, \tilde{x}) = 0$. Since $\tilde{d}_{E'}^\theta(\tilde{x}, \tilde{x}') = \tilde{d}_{E'}^\theta(\operatorname{cl}_{GG'} \tilde{x}, \operatorname{cl}_{GG'} \tilde{x}')$ and $\operatorname{cl}_{GG'} \tilde{x} \in \hat{X}_{GG'}$ for any $\tilde{x} \in \tilde{X}_{GG'}$, the triangle inequality follows from the fact that $\tilde{d}_{E'}^\theta$ is a metric on $\hat{X}_{GG'}$.

Finally, it holds that $\tilde{d}_{E'}^\theta(\tilde{x}, \tilde{x}') = 0$ iff $\operatorname{cl}_{GG'} \tilde{x} = \operatorname{cl}_{GG'} \tilde{x}'$, which in turn is equivalent to $\tilde{X}_{GG'}[\tilde{x}] = \tilde{X}_{GG'}[\tilde{x}']$, by Theorem 3.3. This proves property (3.26).

□

By the one-to-one relation between decompositions and multicuts (Lemma 3.2), every partial characterization of a lifted multicut $\tilde{x} \in \tilde{X}_{GG'}$ defines a class of decompositions of the graph G, namely those defined by the lifted multicuts characterized by $X_{GG'}[\tilde{x}]$. By Theorem 3.6, $\tilde{d}_{E'}^\theta$ with $\theta \in [\frac{1}{2}, 1]$ well-defines a metric on these classes of decompositions and hence a means of comparing the classes of decompositions definable by must-join and must-cut constraints. Computing $\tilde{d}_{E'}^\theta(x, x')$ involves computing the closures of x and x' and is therefore NP-hard (by Theorem 3.4).

3.6 POLYHEDRAL OPTIMIZATION

As the third and final application of lifted multicuts, we turn to the optimization of graph decompositions by lifted multicuts of minimum cost. We start with repeating the definitions of minimum cost multicut and lifted multicut problems.

Definition 2.1. (MC) *For any connected graph $G = (V, E)$, and any $c : E \to \mathbb{R}$, the instance of the* minimum cost multicut problem *w.r.t. G and c is the optimization problem*

$$\min_{x \in X_G} \sum_{e \in E} c_e x_e \, . \tag{MC}$$

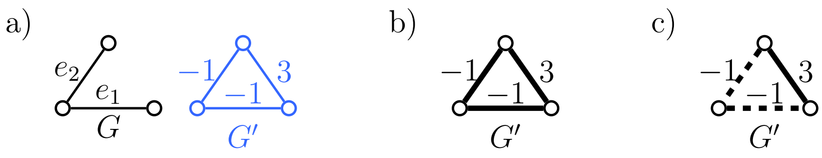


Figure 3.8: Depicted in (a) are two graphs G and G' with costs $c = (-1, -1, 3)$. Depicted in (b) is the optimal solution of the minimum cost lifted multicut problem (Definition 2.3) w.r.t. graphs G, G'. Here, the cost 3 attributed to the additional edge in G' results in the edges e_1 and e_2 not being cut in the optimum $(0, 0, 0)$ which has cost 0. Depicted in (c) is the optimal solution of minimum cost multicut w.r.t. graph G' (Definition 2.1). Here, the cost 3 does not prevent the edges e_1 and e_2 from being cut in the optimum $(1, 1, 0)$ which has cost -2.

Definition 2.3. (LMC) *Given graph $G = (V, E)$ and lifted graph $G' = (V, E')$ where $E \subseteq E' \subseteq \binom{V}{2}$ and a cost function $c : E' \to \mathbb{R}$, the 0/1 linear program written below is called an instance of the* Minimum Cost Lifted Multicut Problem *w.r.t. G, G', and c.*

$$\min_{x \in X_{GG'}} \sum_{e \in E'} c_e x_e \tag{LMC}$$

If $E' = E$, (LMC) specializes to the *minimum cost multicut problem* (MC) w.r.t. G' and c. If $E' \supset E$, the minimum cost lifted multicut problem w.r.t. G, G' and c differs from the minimum cost multicut problem w.r.t. G' and c. It has a smaller feasible set $X_{GG'} \subset X_{G'}$, as we have shown in Section 3.3 and depicted for the smallest example in Figures 3.1 and 3.2. Unlike the minimum cost multicut problem w.r.t. G' and c, the minimum cost lifted multicut problem w.r.t. G, G' and c is such that any feasible solution $x \in X_{GG'}$ indicates by $x_{vw} = 0$ that the nodes v and w are connected in G by a path of edges labeled 0. See also Figure 3.8. This property can be used to penalize by $c_{vw} > 0$ precisely those decompositions of G for which v and w are in distinct components. For nodes v and w that are not neighbors in G, such costs are sometimes called *non-local attractive*.

In order to solve instances of the APX-hard minimum cost lifted multicut problem by means of a branch and cut algorithm, it is useful to study the geometry of lifted multicut polytopes.

Definition 3.9. *(Deza and Laurent, 1997) For any graph $G = (V, E)$, the convex hull $\Xi_G := \operatorname{conv} X_G$ of X_G in $\mathbb{R}^E$ is called the* multicut polytope *of G.*

Definition 3.10. *For any connected graph $G = (V, E)$ and any graph $G' = (V, E')$ with $E \subseteq E'$, $\Xi_{GG'} := \operatorname{conv} X_{GG'}$ is called the* lifted multicut polytope *w.r.t. G and G'.*

Examples are shown in Figures 3.1 and 3.2, respectively. In general, the lifted multicut polytope $\Xi_{GG'}$ w.r.t. graphs G and G' (Figure 3.2) is a subset of the multicut polytope $\Xi_{G'}$ of the graph G' (Figure 3.1). By Lemma 3.4, the system of cycle inequalities (2.1) for G' and cut inequalities (3.5) for G and G' is redundant as a description of $X_{GG'}$ and thus of $\Xi_{GG'}$. Below, we study the geometry of $\Xi_{GG'}$.

3.6.1 DIMENSION

Theorem 3.7. *For any connected graph $G = (V, E)$ and any graph $G' = (V, E')$ with $E \subseteq E'$, $\mathsf{d}\Xi_{GG'} = |E'|$.*

We prove Theorem 3.7 by constructing $|E'| + 1$ multicuts of G' lifted from G whose characteristic functions are affine independent points. The strategy is to construct, for any $e \in E'$, an $x \in X_{GG'}$ with $x_e = 0$ and "as many ones as possible". The challenge is that edges cannot be labeled independently. In particular, for $f \in F_{GG'}$, $x_f = 0$ can imply, for certain $f' \in F_{GG'} \setminus \{f\}$, that $x_{f'} = 0$, as illustrated in Figure 3.9. This structure is made explicit below, in Definition 3.11 and 3.12 and Lemmata 3.11 and 3.12.

Definition 3.11. *For any connected graph $G = (V, E)$ and any graph $G' = (V, E')$ such that $E \subseteq E'$, the sequence $(F_n)_{n \in \mathbb{N}}$ of subsets of $F_{GG'}$ defined below is called the* hierarchy *of $F_{GG'}$ with respect to G:*

1. *$F_0 = \emptyset$*
2. *For any $n \in \mathbb{N}$ and any $\{v, w\} = f \in F_{GG'}$: $\{v, w\} \in F_n$ iff there exists a vw-path in G such that, for any distinct nodes v' and w' in the path such that $\{v', w'\} \neq \{v, w\}$, either $\{v', w'\} \notin F_{GG'}$ or there exists a natural number $j < n$ such that $\{v', w'\} \in F_j$.*

Lemma 3.11. *For any connected graph $G = (V, E)$, any graph $G' = (V, E')$ with $E \subseteq E'$ and any $f \in F_{GG'}$, there exists an $n \in \mathbb{N}$ such that $f \in F_n$.*

Proof. Let $\{v, w\} = f \in F_{GG'}$ and let $d(v, w)$ the length of a shortest vw-path in G. Then, $d(v, w) > 1$ because $F_{GG'} \cap E = \emptyset$.

If $d(v, w) = 2$, there exists a $u \in V$ such that $\{v, u\} \in E$ and $\{u, w\} \in E$. Moreover, $\{v, u\} \notin F_{GG'}$ and $\{u, w\} \notin F_{GG'}$, as $F_{GG'} \cap E = \emptyset$. Thus, $f \in F_1$.

If $d(v, w) = m$ with $m > 2$, consider any shortest vw-path P in G. Moreover, let $F' \subseteq F_{GG'}$ such that, for any $\{v', w'\} = f' \in F_{GG'}$, $f' \in F'$ iff $v' \in P$ and $w' \in P$ and $f' \neq f$. If $F' = \emptyset$ then $f \in F_1$. Otherwise:

$$\forall \{v', w'\} \in F' : \quad d(v', w') < m \tag{3.28}$$

and thus:

$$\forall f' \in F' \; \exists n_{f'} \in \mathbb{N} : \quad f' \in F_{n_{f'}} \tag{3.29}$$

by induction (over m). Let

$$n = \max_{f' \in F'} n_{f'} \, . \tag{3.30}$$

Then, $f \in F_{n+1}$.

□

Definition 3.12. *For any connected graph $G = (V, E)$ and any graph $G' = (V, E')$ with $E \subseteq E'$, the map $\ell : F_{GG'} \to \mathbb{N}$ such that $\forall f \in F_{GG'} \forall n \in \mathbb{N} : \ell(f) = n \Leftrightarrow f \in F_n \wedge f \notin F_{n-1}$ is called the* level function *of $F_{GG'}$.*

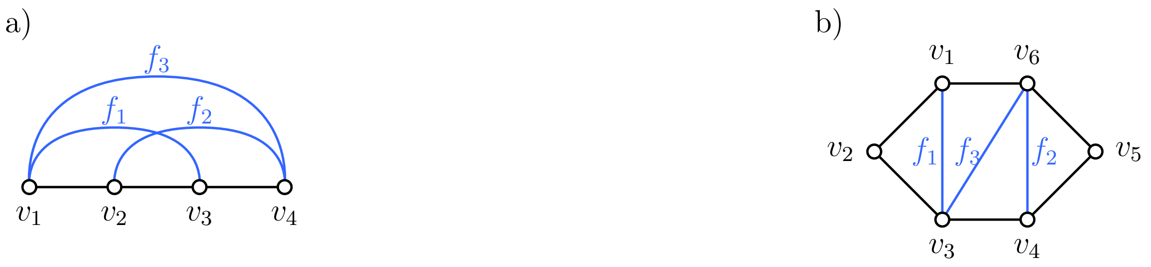


Figure 3.9: If two nodes $\{v, w\} = f \in F_{GG'}$ are in the same component, as indicated by $x_f = 0$, this can imply $x_{f'} = 0$ for one or more $f' \in F \setminus \{f\}$. In (a) $x_{f_3} = 0$ implies $x_{f_1} = 0$ and $x_{f_2} = 0$. In (b) $x_{f_3} = 0$ implies $x_{f_1} = 0$ or $x_{f_2} = 0$.

Lemma 3.12. *For any connected graph $G = (V, E)$, any graph $G' = (V, E')$ with $E \subseteq E'$ and for any $f \in F_{GG'}$, there exists an $x \in X_{GG'}$, called* f-feasible, *such that*

1. $x_f = 0$

2. $x_{f'} = 1$ for all $f' \in F_{GG'} \setminus \{f\}$ with $\ell(f') \geq \ell(f)$.

Proof. For any $\{v, w\} = f \in F_{GG'}$, let P be a shortest vw-path in G and let

$$F'_{GG'} := \{\{v', w'\} \in F_{GG'} \mid v' \in P \wedge w' \in P\} \tag{3.31}$$
$$F''_{GG'} := F_{GG'} \setminus F'_{GG'} \,. \tag{3.32}$$

Moreover, let $x \in \{0,1\}^{E'}$ with $x_P = 0$ and $x_{E \setminus P} = 1$ and $x_{F'_{GG'}} = 0$ and $x_{F''_{GG'}} = 1$. P has no chord in E, because it is a shortest path. Thus, $x \in X_{GG'}$.

□

Proof of Theorem 3.7. The all-one vector $\mathbb{1} \in \{0,1\}^{E'}$ is such that $\mathbb{1} \in X_{GG'}$.

For any $e \in E$, $x^e \in \{0,1\}^{E'}$ such that $x^e_e = 0$ and $x^e_{E \setminus \{e\}} = 1$ and $x^e_{F_{GG'}} = 1$ holds $x^e \in X_{GG'}$.

For any $f \in F_{GG'}$, any f-feasible $x^f \in \{0,1\}^{E'}$ is such that $x^f \in X_{GG'}$. Moreover, x^f can be chosen such that one shortest path connecting the two nodes in f is the only component containing more than one node.

For any $e \in E$, let $y^e \in \mathbb{R}^{E'}$ such that

$$y^e = \mathbb{1} - x^e \,. \tag{3.33}$$

For any $f \in F_1$, choose an f-feasible x^f and let $y^f \in \mathbb{R}^{E'}$ such that

$$y^f = \mathbb{1} - x^f - \sum_{\{e \in E | x^f_e = 0\}} y^e \,. \tag{3.34}$$

For any $n \in \mathbb{N}$ such that $n > 1$ and any $f \in F_n$, choose an f-feasible x^f and let $y^f \in \mathbb{R}^{E'}$ such that

$$y^f = \mathbb{1} - x^f \quad - \sum_{\{f' \in F_{GG'} | f' \neq f \wedge x^f_{f'} = 0\}} y^{f'} \qquad - \sum_{\{e \in E | x^f_e = 0\}} y^e \,. \tag{3.35}$$

Here, $\ell(f') < \ell(f) \leq n$, by definition of f-feasibility. Thus, all $y^{f'}$ are well-defined by induction (over n).

Observe that $\{y^e \mid e \in E'\}$ is the unit basis in $\mathbb{R}^{E'}$. Moreover, each of its elements is a linear combination of $\{\mathbb{1} - x^e \mid e \in E'\}$ which is therefore linearly independent.

Thus, $\{\mathbb{1}\} \cup \{x^e \mid e \in E'\}$ is affine independent. It is also a subset of $X_{GG'}$ and, therefore, a subset of $\Xi_{GG'}$. Thus, $\mathsf{d}\Xi_{GG'} = |E'|$. □

3.6.2 FACETS

Next, we study the facets of $\Xi_{GG'}$. In particular, we consider the facets defined by cut inequalities (3.5). Facets corresponding to other types of inequalities are studied in Horňáková *et al.* (2017). Examples of cuts that are not facet-defining for $\Xi_{GG'}$ are shown in Figure 3.11. To constrain the class of cuts that are facet-defining, we introduce additional notation: For any connected graph $G = (V, E)$, any distinct nodes $v, w \in V$ and any $C \in vw\text{-cuts}(G)$, we denote by

$$G(v, C) = (V(v, C), E(v, C)), \quad G(w, C) = (V(w, C), E(w, C)) \tag{3.36}$$

the largest components of the graph $(V, E \setminus C)$ that contain v and w, respectively. By definition of a vw-cut[5], we have

$$V(v, C) \cap V(w, C) = \emptyset \wedge \quad V(v, C) \cup V(w, C) = V\,. \tag{3.37}$$

We denote by $F_{GG'}(vw, C)$ the set of those edges in $F_{GG'}$, except vw, that cross the vw-cut C of G, i.e.

$$F_{GG'}(vw, C) := \{f \in F_{GG'} \setminus \{vw\} \mid f \not\subseteq V(v, C) \wedge f \not\subseteq V(w, C)\}\,. \tag{3.38}$$

We denote by $G'(vw, C) := (V, F_{GG'}(vw, C) \cup C)$ the subgraph of G' that comprises all edges from $F_{GG'}(vw, C)$ and C. Finally, we define

$$S_{GG'}(vw, C) := \left\{ x \in X_{GG'} \,\middle|\, 1 - x_{vw} = \sum_{e \in C} (1 - x_e) \right\} \tag{3.39}$$

$$\Sigma_{GG'}(vw, C) := \operatorname{conv} S_{GG'}(vw, C)\,. \tag{3.40}$$

Definition 3.13. *For any connected graph $G = (V, E)$, any distinct $v, w \in V$ and any $C \in vw$-cuts(G), a component (V^*, E^*) of G is called* properly (vw, C)-connected *iff*

$$v \in V^* \;\wedge\; w \in V^* \;\wedge\; |E^* \cap C| = 1\,. \tag{3.41}$$

It is called improperly (vw, C)-connected *iff*

$$V^* \subseteq V(v, C) \;\vee\; V^* \subseteq V(w, C)\,. \tag{3.42}$$

It is called (vw, C)-connected *iff it is properly or improperly (vw, C)-connected.*

[5]For any graph $G = (V, E)$ and any distinct nodes $v, w \in V$, a vw-cut of G is a minimal (with respect to inclusion) set $C \subseteq E$ such that v and w are not connected in $(V, E \setminus C)$.

For any (vw, C)-connected component (V^*, E^*) of G, we denote by $F_{V^*} := \{v'w' = f' \in F_{GG'}(vw, C) \mid v' \in V^* \wedge w' \in V^*\}$ the set of those edges $v'w' = f' \in F_{GG'}(vw, C)$ such that (V^*, E^*) is also $(v'w', C)$-connected.

Proposition 3.1. *For every connected graph $G = (V, E)$, every graph $G' = (V, E')$ with $E \subseteq E'$, every $vw \in F_{GG'}$ and every $C \in vw$-cuts(G), the following holds:*

1. *Every $x \in S_{GG'}(vw, C)$ defines a decomposition of G into (vw, C)-connected components. That is, every maximal component of the graph $(V, \{e \in E | x_e = 0\})$ is (vw, C)-connected. At most one of these is properly (vw, C)-connected. It exists iff $x_{vw} = 0$.*
2. *For every (vw, C)-connected component (V^*, E^*) of G, the $x \in \{0, 1\}^{E'}$ such that $\forall rs \in E'(x_{rs} = 0 \Leftrightarrow r \in V^* \wedge s \in V^*)$ is such that $x \in S_{GG'}(vw, C)$.*

Proof. 1. Let $x \in S_{GG'}(vw, C)$ arbitrary. Let $E_0 := \{e \in E | x_e = 0\}$ and let $G_0 := (V, E_0)$.

If $x_{vw} = 1$ then $\forall e \in C : x_e = 1$, by (3.39). Thus, every component of G_0 is improperly (vw, C)-connected.

If $x_{vw} = 0$ then

$$\exists e \in C(x_e = 0 \wedge \forall e' \in C \setminus \{e\}(x_{e'} = 1)) \tag{3.43}$$

by (3.39). Let (V^*, E^*) the maximal component of G_0 with

$$e \in E^* . \tag{3.44}$$

Clearly:

$$\forall e' \in C \setminus \{e\} : \ e' \notin E^* \tag{3.45}$$

by (3.43) and definition of G_0. There does not exist a $C' \in vw$-cuts(G) with $x_{C'} = 1$, because this would imply $x_{vw} = 1$, by (3.5). Thus, there exists a $P \in vw$-paths(G) with $x_P = 0$, as G is connected. Any such path P has $e \in P$, as $P \cap C \neq \emptyset$ and $C \cap E_0 = \{e\}$ and $P \subseteq E_0$. Thus:

$$v \in V^* \wedge w \in V^* \tag{3.46}$$

by (3.44). (V^*, E^*) is properly (vw, C)-connected, by (3.44), (3.45) and (3.46). Any other component of G_0 does not cross the cut, by (3.43), (3.44) and definition of G_0, and is thus improperly (vw, C)-connected.

2. We have

$$\forall st \in E : \ x_{st} = 0 \Leftrightarrow st \in E^* \tag{3.47}$$

by the following argument:

- If $st \in E^*$, then $s \in V^* \wedge t \in V^*$, as (V^*, E^*) is a graph. Thus, $x_{st} = 0$, by definition of x.

- If $st \notin E^*$ then $s \notin V^* \vee t \notin V^*$, as (V^*, E^*) is a component of G. Thus, $x_{st} = 1$, by definition of x.

Consider the decomposition of G into (V^*, E^*) and singleton components. $E_1 := \{e \in E | x_e = 1\}$ is the set of edges that straddle distinct components of this decomposition, by (3.47). Therefore, E_1 is a multicut of G, by Lemma 3.2. Thus, (3.3) holds.

For any $st \in F_{GG'}$ and any $P \in st\text{-paths}(G)$, distinguish two cases:

- If $P \subseteq E^*$, then $s \in V^* \wedge t \in V^*$, as (V^*, E^*) is a graph. Thus, $x_{st} = 0$, by definition of x. Moreover, $x_P = 0$, by (3.47). Hence, (3.4) evaluates to $0 = 0$.
- Otherwise, there exists an $e \in P$ such that $e \notin E^*$. Therefore, $x_e = 1$, by (3.47). Thus, (3.4) holds, as the r.h.s. is at least 1.

For any $st \in F_{GG'}$ and any $C' \in st\text{-cuts}(G)$, distinguish two cases:

- If $C' \cap E^* = \emptyset$ then $s \notin V^* \vee t \notin V^*$. Therefore, $x_{st} = 1$, by definition of x. Moreover, $x_{C'} = 1$, by (3.47). Thus, (3.5) evaluates to $0 = 0$.
- Otherwise, there exists an $e \in C'$ such that $e \in E^*$. Therefore, $x_e = 0$, by (3.47). Thus, (3.5) holds, as the r.h.s. is at least 1.

□

Theorem 3.8. *For any connected graph $G = (V, E)$, any graph $G' = (V, E')$ with $E \subseteq E'$, any $vw = f \in F_{GG'}$ and any $C \in vw\text{-cuts}(G)$, $\Sigma_{GG'}(vw, C)$ is a facet of $\Xi_{GG'}$ only if the following necessary conditions hold:*

1. *For any $e \in C$, there exists a (vw, C)-connected component (V^*, E^*) of G such that $e \in E^*$.*

2. *For any $\emptyset \neq F \subseteq F_{GG'}(vw, C)$, there exists an edge $e \in C$ and (vw, C)-connected components (V^*, E^*) and (V^{**}, E^{**}) of G such that $e \in E^*$ and $e \in E^{**}$ and $|F \cap F_{V^*}| \neq |F \cap F_{V^{**}}|$.*

3. *For any $f' \in F_{GG'}(vw, C)$, any $\emptyset \neq F \subseteq F_{GG'}(vw, C) \setminus \{f'\}$ and any $k \in \mathbb{N}$, there exist (vw, C)-connected components (V^*, E^*) and (V^{**}, E^{**}) with $f' \in F_{V^*}$ and $f' \notin F_{V^{**}}$ such that*

$$|F \cap F_{V^*}| \neq k \text{ or } |F \cap F_{V^{**}}| \neq 0 \,. \tag{3.48}$$

4. *For any $v' \in V(v, C)$, any $w' \in V(w, C)$ and any $v'w'$-path $P = (V_P, E_P)$ in $G'(vw, C)$, there exists a properly (vw, C)-connected component (V^*, E^*) of G such that*

$$\begin{aligned} &(v' \notin V^* \vee \exists w'' \in V_P \cap V(w, C) : w'' \notin V^*) \\ &\wedge (w' \notin V^* \vee \exists v'' \in V_P \cap V(v, C) : v'' \notin V^*) \,. \end{aligned} \tag{3.49}$$

5. *For any cycle $Y = (V_Y, E_Y)$ in $G'(vw, C)$, there exists a properly (vw, C)-connected component (V^*, E^*) of G such that*

$$(\exists v' \in V_Y \cap V(v, C) : v' \notin V^*) \wedge (\exists w' \in V_Y \cap V(w, C) : w' \notin V^*) \,. \tag{3.50}$$

Proof. 1. Assume that that Condition 1 does not hold (as in Figure 3.11a). Then, there exists an $e \in C$ such that no (vw, C)-connected component of G contains e. Thus, for all $x \in S_{GG'}(vw, C)$:

$$x_e = 1 \tag{3.51}$$

by Proposition 3.1. Now, $\mathsf{d}\Sigma_{GG'}(vw, C) \leq |E'| - 2$, by (3.39) and (3.51). Thus, $\Sigma_{GG'}(vw, C)$ is not a facet of $\Xi_{GG'}$, by Theorem 3.7.

2. Assume that Condition 2 does not hold. Then, for any $e \in C$ there exists some number m such that for all (vw, C)-connected components (V^*, E^*) with $e \in E^*$ it holds that $|F \cap F_{V^*}| = m$. Thus, we can write

$$C = \bigcup_{m=0}^{|F|} C(F, m), \tag{3.52}$$

where $C(F, m) := \big\{e \in C \mid |F \cap F_{V^*}| = m \ \forall\ (vw, C)\text{-connected } (V^*, E^*) \text{ with } e \in E^*\big\}$. It follows that for all $x \in S_{GG'}(vw, C)$ we have the equality

$$\sum_{m=0}^{|F|} m \sum_{e \in C(F,m)} (1 - x_e) = \sum_{f' \in F} (1 - x_{f'}) \tag{3.53}$$

by the following argument:

- If $x_e = 1$ for all $e \in C$, then $x_{f'} = 1$ for all $v'w' = f' \in F$, since C is also a $v'w'$-cut. Thus, (3.53) evaluates to $0 = 0$.
- Otherwise there exists precisely one edge $e \in C$ such that $x_e = 0$. Let m be such that $e \in C(F, m)$. By definition of $C(F, m)$, there are exactly m edges $f' \in F$ with $x_{f'} = 0$. Thus, (3.53) evaluates to $m = m$.

3. Assume that Condition 3 does not hold. Then there exists an $f' \in F_{GG'}(vw, C)$, a set $\emptyset \neq F \subseteq F_{GG'}(vw, C)$ and some $k \in \mathbb{N}$ such that for all (vw, C) connected components (V^*, E^*) and (V^{**}, E^{**}) with $f' \in F_{V^*}$ and $f' \notin F_{V^{**}}$ it holds that

$$|F \cap F_{V^*}| = k \text{ and } |F \cap F_{V^{**}}| = 0. \tag{3.54}$$

In other words, for all $x \in S_{GG'}(vw, C)$ it holds that $x_{f'} = 0$ iff there are exactly k edges $f'' \in F$ such that $x_{f''} = 0$. Similarly, it holds that $x_{f'} = 1$ iff for all $f'' \in F$ we have $x_{f''} = 1$. Therefore, all $x \in S_{GG'}(vw, C)$ satisfy the additional equality

$$k(1 - x_{f'}) = \sum_{f'' \in F} 1 - x_{f''}. \tag{3.55}$$

4. Assume that Condition 4 does not hold. Then, there exist $v' \in V(v, C)$ and $w' \in V(w, C)$ and a $v'w'$-path $P = (V_P, E_P)$ in $G'(vw, C)$ such that every properly (vw, C)-connected component (V^*, E^*) of G holds:

$$(v' \in V^* \ \wedge\ V(w, C) \cap V_P \subseteq V^*) \tag{3.56}$$
$$\vee\, (w' \in V^* \ \wedge\ V(v, C) \cap V_P \subseteq V^*)\,. \tag{3.57}$$

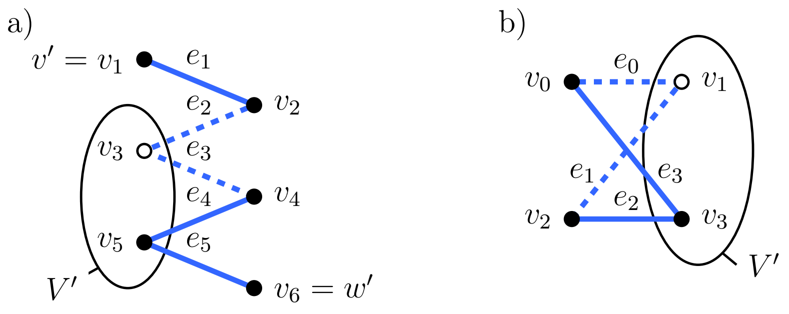


Figure 3.10: Illustrations for the proof of Theorem 3.8. Depicted are the nodes (in black) and edges (in blue) on a path illustrating Condition 4 (a) and on a cycle illustrating Condition 5 (b), respectively. Nodes in the set V' are either in V^* (filled circle) or not in V^* (open circle). Consequently, pairs of consecutive edges are either cut (dashed lines) or not cut (solid lines).

Let $v_1 < \cdots < v_{|V_P|}$ the linear order of the nodes V_P and let $e_1 < \cdots < e_{|E_P|}$ the linear order of the edges E_P in the $v'w'$-path P. Now, for all $x \in S_{GG'}(vw, C)$:

$$x_{vw} = \sum_{j=1}^{|E_P|} (-1)^{j+1} x_{e_j} \tag{3.58}$$

by the following argument: $|E_P|$ is odd, as the path P alternates between the set $V(v, C)$ where it begins and the set $V(w, C)$ where it ends. Thus,

$$\sum_{j=1}^{|E_P|} (-1)^{j+1} x_{e_j} = x_{e_1} - \sum_{j=1}^{(|E_P|-1)/2} (x_{e_{2j}} - x_{e_{2j+1}}) \, . \tag{3.59}$$

Distinguish two cases:

- If $x_{vw} = 1$, then $x_{E_P} = 1$, by (3.39) and (3.5). Thus, (3.58) evaluates to $1 = 1$, by (3.59).
- If $x_{vw} = 0$, the decomposition of G defined by x contains precisely one properly (vw, C)-connected component (V^*, E^*) of G, by Proposition 3.1. Without loss of generality, (3.56) holds. Otherwise, that is, if (3.57) holds, exchange v and w.

  Consider the nodes V_P as depicted in Figure 3.10a: $v_1 = v' \in V^*$, by (3.56). For every even j, $v_j \in V(w, C)$, by definition of P. Thus:

  $$\forall j \in \{1, \ldots, (|E_P| + 1)/2\} : \; v_{2j} \in V^* \tag{3.60}$$

  by (3.56).

  Consider the edges E_P as depicted in Figure 3.10a: $e_1 = v_1 v_2 \in E^*$, as $v_1 \in V^*$ and $v_2 \in V^*$ and as (V^*, E^*) is a component of G. Thus,

  $$x_{e_1} = 0 \tag{3.61}$$

  by Proposition 3.1. For every $j \in \{1, \ldots, (|E_P| - 1)/2\}$, distinguish two cases:

- If $v_{2j+1} \in V^*$, then $e_{2j} = v_{2j}v_{2j+1} \in E^*$ and $e_{2j+1} = v_{2j+1}v_{2j+2} \in E^*$, because $v_{2j} \in V^*$ and $v_{2j+2} \in V^*$, by (3.60), and because (V^*, E^*) is a component of G. Thus:

$$x_{e_{2j}} = 0 \;\wedge\; x_{e_{2j+1}} = 0 \,. \tag{3.62}$$

- If $v_{2j+1} \notin V^*$, then $e_{2j} = v_{2j}v_{2j+1}$ and $e_{2j+1} = v_{2j+1}v_{2j+2}$ straddle distinct components of the decomposition of G defined by x, because $v_{2j} \in V^*$ and $v_{2j+2} \in V^*$, by (3.60). Thus:

$$x_{e_{2j}} = 1 \;\wedge\; x_{e_{2j+1}} = 1 \,. \tag{3.63}$$

In any case:

$$\forall j \in \{1, \dots, (|E_P| - 1)/2\} : \; x_{e_{2j}} - x_{e_{2j+1}} = 0 \,. \tag{3.64}$$

Thus, (3.58) evaluates to $0 = 0$, by (3.59), (3.61), (3.64).

5. Assume that Condition 5 does not hold. Then, there exists a cycle $Y = (V_Y, E_Y)$ in $G'(vw, C)$ such that every properly (vw, C)-connected component (V^*, E^*) of G holds:

$$V_Y \cap V(v, C) \subseteq V^* \tag{3.65}$$

$$\vee \quad V_Y \cap V(w, C) \subseteq V^* \,. \tag{3.66}$$

Let $v_0 < \cdots < v_{|V_Y|-1}$ an order on V_Y such that $v_0 \in V(v, C)$ and, for all $j \in \{0, \dots, |E_Y| - 1\}$:

$$e_j := \{v_j, v_{j+1 \bmod |E_Y|}\} \in E_Y \,. \tag{3.67}$$

Now, for all $x \in S_{GG'}(vw, C)$:

$$0 = \sum_{j=0}^{|E_Y|-1} (-1)^j x_{e_j} \tag{3.68}$$

by the following argument: $|E_Y|$ is even, as the cycle Y alternates between the sets $V(v, C)$ and $V(w, C)$. Thus,

$$\sum_{j=0}^{|E_Y|-1} (-1)^j x_{e_j} = \sum_{j=0}^{(|E_Y|-2)/2} (x_{e_{2j}} - x_{e_{2j+1}}) \,. \tag{3.69}$$

Distinguish two cases:

- If $x_{vw} = 1$, then $x_{E_Y} = 1$, by (3.39) and (3.5). Thus, (3.68) evaluates to $0 = 0$, by (3.69).

- If $x_{vw} = 0$, the decomposition of G defined by x contains precisely one properly (vw, C)-connected component (V^*, E^*) of G, by Proposition 3.1. Without loss of generality, (3.65) holds. Otherwise, that is, if (3.66) holds, exchange v and w.

  Consider the nodes V_Y as depicted in Figure 3.10b: For every even j, $v_j \in V(v, C)$, by definition of Y and the order. Thus:

$$\forall j \in \{0, \dots, (|E_Y| - 2)/2\} : \; v_{2j} \in V^* \tag{3.70}$$

  by (3.65).

  Consider the edges E_Y as depicted in Figure 3.10b: For every $j \in \{0, \dots, (|E_Y| - 2)/2\}$, distinguish two cases:

  - If $v_{2j+1} \in V^*$, then $e_{2j} = v_{2j}v_{2j+1} \in E^*$ and $e_{2j+1} = v_{2j+1}v_{2j+2 \bmod |E_Y|} \in E^*$, because $v_{2j} \in V^*$ and $v_{2j+2 \bmod |E_Y|} \in V^*$, by (3.70), and because (V^*, E^*) is a component of G. Thus:

$$x_{e_{2j}} = 0 \;\wedge\; x_{e_{2j+1}} = 0 \,. \tag{3.71}$$

  - If $v_{2j+1} \notin V^*$, then $e_{2j} = v_{2j}v_{2j+1}$ and $e_{2j+1} = v_{2j+1}v_{2j+2 \bmod |E_Y|}$ straddle distinct components of the decomposition of G defined by x, because $v_{2j} \in V^*$ and $v_{2j+2 \bmod |E_Y|} \in V^*$, by (3.70). Thus:

$$x_{e_{2j}} = 1 \;\wedge\; x_{e_{2j+1}} = 1 \,. \tag{3.72}$$

  In any case:

$$\forall j \in \{0, \dots, (|E_Y| - 2)/2\} : \; x_{e_{2j}} - x_{e_{2j+1}} = 0 \,. \tag{3.73}$$

  Thus, (3.68) evaluates to $0 = 0$, by (3.69) and (3.73).

□

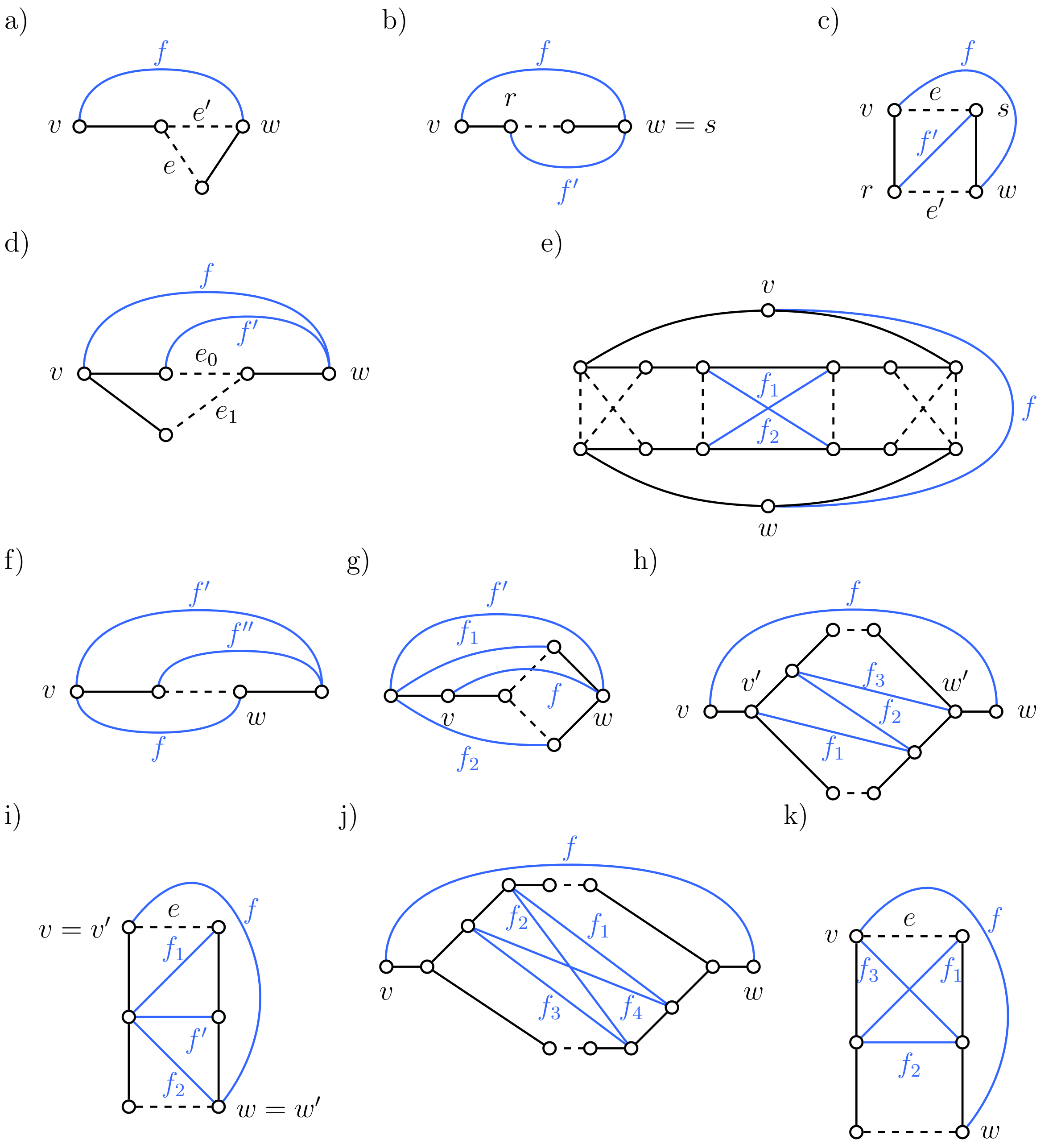


Figure 3.11: Depicted above are graphs $G = (V, E)$ (in black) and $G' = (V, E')$ with $E \subseteq E'$ (E' in blue), distinct nodes $v, w \in V$ and a vw-cut C of G (as dashed lines). In any of the above examples, one condition of Theorem 3.8 is violated and thus, $\Sigma_{GG'}(vw, C)$ is not a facet of the lifted multicut polytope $\Xi_{GG'}$. a) Condition 1 is violated for e. b) Condition 2 is violated as r and s are connected in any (vw, C)-connected component. c) Condition 2 is violated as r and s are not connected in any (vw, C)-connected component. d) Condition 2 is violated. Specifically, $C(\{f'\}, 1) = \{e_0\}$ and $C(\{f'\}, 0) = \{e_1\}$ in the proof of Theorem 3.8. e) Condition 2 is violated for $F = \{f_1, f_2\}$. f) Condition 3 is violated. g) Condition 3 is violated for $F = \{f_1, f_2\}$ and $k = 1$. h) Condition 4 is violated for the $v'w'$-path $f_1 f_2 f_3$. i) Condition 4 is violated for the $v'w'$-path $e f_1 f_2$. j) Condition 5 is violated for the cycle $f_1 f_2 f_3 f_4$. k) Condition 5 is violated for the cycle $e f_1 f_2 f_3$.

4 LIFTED DISJOINT PATHS WITH APPLICATION IN MULTIPLE OBJECT TRACKING

4.1 INTRODUCTION

THE predominant approach for MOT is the tracking-by-detection paradigm, which splits the problem into two subtasks. First, objects are detected in all video frames by an object detector. Then, the detections are linked across frames to form trajectories. While the performance of object detectors has improved considerably by recent advances of CNNs (Ren *et al.*, 2015; Yang *et al.*, 2016; Redmon *et al.*, 2016; Duan *et al.*, 2019), the latter task called the *data association* remains challenging. We concentrate on the latter task in this work.

While for MOT even very large data association instances can be solved using the disjoint paths formulation (DP), it has been shown that the basic disjoint paths problem alone is not sufficient to provide trajectories of high accuracy. The main limitation for MOT is the implicit assumption of a first-order Markov chain. In particular, costs only indicate whether two detections directly follow each other in a track.

Our contribution is three-fold: First, to overcome the limited expressiveness of disjoint paths, we propose to augment it with lifted edges that take into account long-range interactions (see Figure 4.1). We call the resulting problem the *lifted disjoint paths problem* (Section 4.3). We prove the problem to be NP-hard in Section 4.6. Second, we study the optimization problem from a polyhedral perspective, proposing a high-quality linear programming relaxation in Section 4.4 together with efficient separation routines for the proposed constraints (Section 4.5). Third, we apply the lifted disjoint paths problem to MOT. We show in Section 4.7 that our solver LifT significantly outperforms trackers that were state-of-the-art on the popular MOT challenge datasets at the time of publishing our paper Horňáková *et al.* (2020).

We argue that our model has advantages from the modeling and optimization point of view. From the modeling standpoint, the lifted disjoint paths problem does not change the set of feasible solutions but adds more expressive power to it. For MOT, this means that the set of feasible solutions, which naturally represent trajectories of objects, is preserved. The additional lifted edges represent connectivity priors. A lifted edge is active if and only if there is an active trajectory between its endpoints in the flow graph. For MOT, lifted edges take (dis-)similarity of object detection pairs represented by its endpoints into account. This allows to encourage or penalize an active path between the detections with possibly larger temporal distance. This helps to re-identify the same object and to prevent id-switches between distinct objects within long trajectories.

From the optimization point of view, we study several non-trivial classes of linear inequalities that result in a high-quality relaxation. The proposed inequalities depend non-trivially on the constraint structure of the underlying disjoint paths problem (Section 4.4). We show that the polyhedral relaxation that we consider is tighter than naively applying known inequalities. The proposed relaxation enables us to solve MOT problems via a global approach, in contrast to established approaches, which either use heuristics on complex models or global optimization on simpler models that do not exploit long-range interaction. We present, to our knowledge, the first global optimization approach that incorporates long-range interaction for MOT. This has several advantages: First, our optimization is not trapped in poor local optima or affected by initialization choices and is hence potentially more robust. Second, improvements in the discriminative power of features used to compute costs for the lifted disjoint paths problem directly correlate to better tracking performance, since no errors are introduced by suboptimal choices during optimization.

Finally, we note that the proposed lifted disjoint paths formulation is not inherently tied to MOT and can potentially be applied to further problems not related to MOT.

Our code is available at https://github.com/AndreaHor/LifT_Solver.

4.2 RELATED WORK

Extended disjoint paths for MOT. We discuss the usage of (DP) for MOT in Section 2.3.3. Extensions of the plain disjoint paths problem that disallow certain pairs of detections to occur simultaneously have been used to fuse different object detectors by Chari *et al.* (2015) and for multi-camera MOT by Hofmann *et al.* (2013) and Leal-Taixé *et al.* (2012). Jiang *et al.* (2007) extended (DP) by simultaneously minimizing not only costs between consecutive detections within object trajectories but also changes in mutual positions of object pairs in consecutive frames. The drawback of these approaches is that they cannot integrate long-range information, in contrast to our proposed formulation.

Other combinatorial approaches to MOT. We discussed the usage of multicut and lifted multicut models for MOT in Sections 2.1.3 and 2.2.2. Some methods employed bipartite matching (Sadeghian *et al.*, 2017; Zhu *et al.*, 2018; Xu *et al.*, 2019; Wojke *et al.*, 2017) to optimally assign new detections to already computed trajectories in the past. Arora and Globerson (2013) used a formulation equivalent to the 3-matching problem where each triplet of detections in three consecutive frames is assigned a cost that is paid iff the detections are connected. The minimum cost arborescence problem, an extension of the minimum spanning tree to directed graphs, has been used for MOT by Henschel *et al.* (2014). Zamir *et al.* (2012) applied multiple times generalized minimum clique problem (minimum cost clique in a complete K-partite graph) to obtain the trajectory of one object in each iteration. Dehghan *et al.* (2015) formulated data association as a generalized maximum multi clique problem that enabled finding the trajectories jointly for all objects. The maximum weight independent set has been used for data association by Brendel *et al.* (2011). Hu *et al.* (2019) formulated data association as a multi-dimensional assignment problem. The works by Henschel *et al.* (2018, 2016) reformulated the tracking of multiple objects with long temporal interactions as a binary quadratic program. If the problem size is small, the optimization problem can be solved optimally by reformulating it to an equivalent binary linear program (Henschel *et al.*, 2019a; von Marcard *et al.*, 2018). For large instances, an approximation is necessary. To this end, a specialized non-convex Frank-Wolfe method can be used (Henschel *et al.*, 2018). Common to the above state-of-the-art trackers is that they either employed heuristic solvers or are limited in the integration of long-range information, in contrast to our work.

Contribution w.r.t. existing combinatorial approaches. It is widely acknowledged that one crucial ingredient for obtaining high-quality MOT results is to incorporate long-range temporal information to re-identify detections and prevent id-switches. However, from a theoretical perspective, we believe that long-range information has not yet been incorporated satisfactorily in optimization formulations for the data association step in MOT.

In comparison to lifted multicut for MOT, we argue that from the modelling point of view, network flow has advantages. In multicut, clusters can be arbitrary, while in MOT, tracks are clusters that cannot contain multiple object detections at the same time point. This exclusion constraint must be enforced in multicut explicitly via soft constraints, while the disjoint paths substructure automatically takes care of it. On the other hand, the lifted multicut approach (Tang *et al.*, 2017) has used the possibility to cluster multiple detections in one time frame. This directly incorporates non-maxima suppression in the optimization, which however increases computational complexity.

From a mathematical perspective, naively using polyhedral results from multicut is also not satisfactory. Specifically, one could naively obtain a polyhedral relaxation for the lifted disjoint paths problem by reusing lifted multicut Constraints (3.3)-(3.5) and additionally adding network flow constraints for the disjoint paths substructure. However, this would give a suboptimal polyhedral relaxation. We show in Section 4.4 that the underlying structure of the disjoint paths problem can be used to derive new and tighter constraints for lifted edges. This enables us to use a global optimization approach for MOT. To our knowledge, our work is the first one to combine global optimization with long-range interactions for MOT.

In comparison to works that proposed non-convex algorithms or other heuristics for incorporating long-range temporal edges (Henschel *et al.*, 2018; Hu *et al.*, 2019; Zamir *et al.*, 2012; Dehghan *et al.*, 2015) our approach yields a more principled approach and globally optimal optimization solutions via LP-based branch and cut algorithms.

4.3 PROBLEM FORMULATION

Below, we extend the disjoint paths problem by defining lifted edges. We discuss how the lifted disjoint paths problem can naturally model MOT.

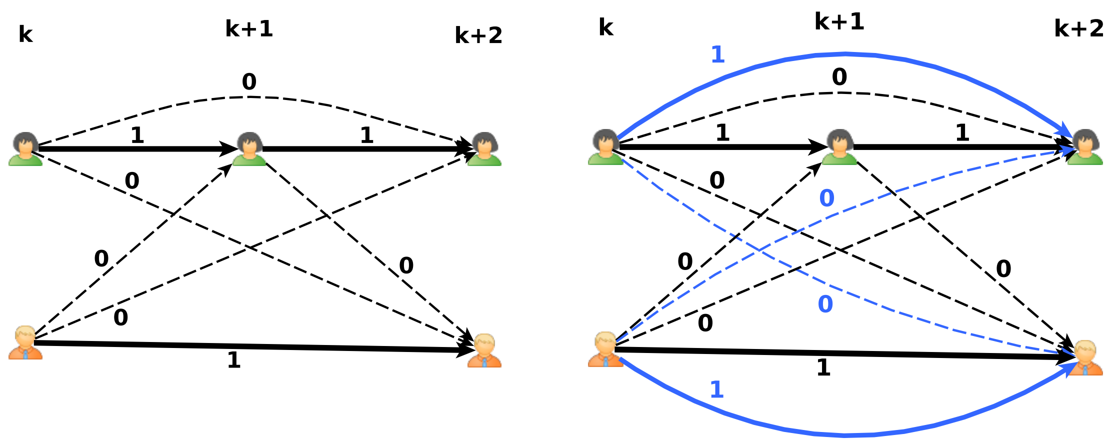


Figure 4.1: Depicted left is an illustration of disjoint paths problem (DP). Depicted right is its extension to lifted disjoint paths problem (LDP). Lifted edges are depicted in blue. The three graph layers represent three consecutive time frames. Dashed edges labeled by zero represent zero flow, resp. inactive lifted edges. Thick edges labeled by one represent unit flow, resp. active lifted edges.

Flow network and lifted graph. Consider two directed acyclic graphs $G = (V, E)$ and $G' = (V', E')$ where $V' = V \backslash \{s, t\}$. The graph $G = (V, E)$ represents the *flow network* and we denote by G' the lifted graph. The two special nodes s and t of G denote source and sink node respectively. We further assume that every node in V is reachable from s, and t can be reached from it.

We define the set of paths starting at v and ending in w as

$$vw\text{-paths}(G) = \left\{ (v_1v_2, \dots, v_{l-1}v_l) : \begin{array}{c} v_iv_{i+1} \in E, \\ v_1 = v, v_l = w \end{array} \right\} . \tag{4.1}$$

For a vw-path P we denote its edge set as P_E and its node set as P_V.

The flow variables in G are denoted by $y \in \{0,1\}^E$ for edges and $z \in \{0,1\}^V$ for nodes. Allowing only $0/1$ values of vertex variables reflects the requirement of vertex disjoint paths. Variables on the lifted edges E' are denoted by $y' \in \{0,1\}^{E'}$. Here, $y'_{vw} = 1$ means that nodes v and w are connected via the flow y in G. Formally,

$$y'_{vw} = 1 \Leftrightarrow \exists P \in vw\text{-paths}(G) \text{ s.t. } \forall ij \in P_E : y_{ij} = 1 \, . \tag{4.2}$$

Definition 4.1. (LDP) *Given edge costs $c \in \mathbb{R}^E$, node cost $d \in \mathbb{R}^V$ in flow network G and edge cost $c' \in \mathbb{R}^{E'}$ for the lifted graph G' we define the* lifted disjoint paths problem *as*

$$\begin{aligned} \min_{\substack{y \in \{0,1\}^E, y' \in \{0,1\}^{E'}, \\ z \in \{0,1\}^V}} \quad & \langle c, y \rangle + \langle c', y' \rangle + \langle d, z \rangle \\ s.t. \quad & y \in Y^{DP}(G, s, t) \\ & z \textit{ flow through nodes of } G \textit{ according to } (2.7) \\ & y, y' \textit{ feasible according to } (4.2) \end{aligned} \tag{LDP}$$

In Section 4.4, we present an ILP formulation of (LDP) by proposing several linear inequalities that lead to a high-quality linear relaxation.

Graph construction for multiple object tracking. We argue that the lifted disjoint paths problem is an appropriate way of modelling the data association problem for MOT. In MOT, an unknown number of objects needs to be tracked across a video sequence. This problem can be naturally formalized by a graph $G = (V, E)$ where its node set V represents either object detections or tracklets of objects. If V represents object detections, we can express it as follows: $V = s \cup V_1 \cup \ldots \cup V_T \cup t$, where T is the number of frames and V_i represents the object detections in time i. We introduce edges between adjacent time frames. An active flow on such an edge denotes correspondences of the same object. We also introduce skip edges between time frames that are farther apart. An active flow on a skip edge also denotes correspondences between the same object that, in contrast, may have been occluded or not detected in the intermediate time frames. This classical network flow formulation has been commonly used for MOT (Zhang *et al.*, 2008).

On top of the underlying flow formulation for MOT, we want to express that two detections belong to the same object connected by a possibly longer track with multiple detections in between. For that purpose, lifted edges with negative costs can be used. We say in such a case that an active lifted edge re-identifies two detections (Tang *et al.*, 2017). If two detections with a larger temporal distance should not be part of the same track, a positive valued lifted edge can be used. In this case, the lifted edge is used to prevent id-switches.

4.4 CONSTRAINTS

Below, we will first introduce constraints that give an integer linear program (ILP) of the lifted disjoint paths problem (LDP). The corresponding LP relaxation can be tightened by additional constraints that we present subsequently.

Many constraints considered below will rely on whether a node w is reachable from another node v in the flow network $G = (V, E)$. We define to this end the *reachability relation* $\mathcal{R}_G \subset V^2$ via

$$vw \in \mathcal{R}_G \Leftrightarrow vw\text{-paths}(G) \neq \emptyset\,. \tag{4.3}$$

In the special case of $v = w$, we also allow empty paths, which means $\forall v \in V : vv \in \mathcal{R}_G$. This makes relation $\mathcal{R}_G$ reflexive.

Flow conservation constraints. The flow variables y obey, as in classical network flow problems (Ahuja *et al.*, 1988), the flow conservation constraints

$$\forall v \in V \setminus \{s, t\} : \sum_{uv \in E} y_{uv} = \sum_{vw \in E} y_{vw} = z_v\,. \tag{4.4}$$

Constraining lifted edges. All the following constraints restrict the values of lifted edge variables y'_{vw} in order to ensure that they satisfy (4.2). Despite their sometimes complex form, they always obey the two basic principles:

- If there is flow in G going from vertex v to vertex w, then $y'_{vw} = 1$. The constraints of this form are (4.7), (4.13).

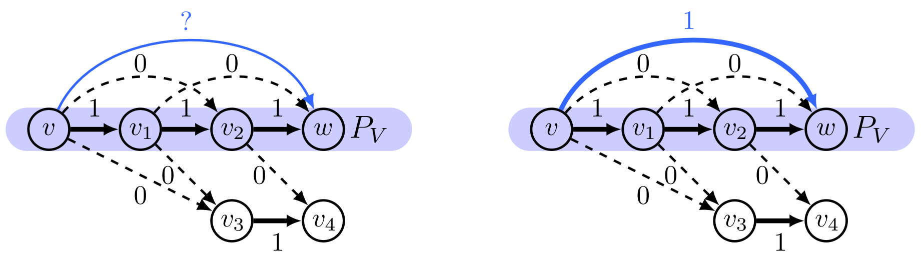


Figure 4.2: An illustration of the path inequalities (4.7). Given the active path $P = (vv_1, v_1v_2, v_2w)$, the label of lifted edge vw (labeled with a question mark in the left figure) is determined by inequality $y'_{vw} \geq y_{vv_1} + y_{vv_2} - y_{v_1v_3} - y_{v_2v_4} = 1 + 0 - 0 - 0 = 1$ to be one (right figure). Note that even if the flow skipped vertex v_1 or v_2, this inequality would still give the correct bound $y'_{vw} \geq 1$.

- If there is a vw-cut in G with all edges labeled by zero (i.e. no flow passes through this cut), then $y'_{vw} = 0$. We will mainly look at cuts that are induced by paths, i.e. edges that separate a path from the rest of the graph. The paths of interest will either originate at v or end at w. The constraints of this form are (4.5), (4.6), (4.12), (4.17), (4.18).

Single node cut inequalities. Given a lifted edge $vw \in E'$, if there is no flow going from vertex v which can potentially go to vertex w, then $y'_{vw} = 0$. Formally,

$$y'_{vw} \leq \sum_{\substack{u:\ vu \in E, \\ uw \in \mathcal{R}_G}} y_{vu}\,. \tag{4.5}$$

Similarly, if there is no flow going to w that can originate from vertex v, then $y'_{vw} = 0$. Formally,

$$y'_{vw} \leq \sum_{\substack{u: uw \in E, \\ vu \in \mathcal{R}_G}} y_{uw}\,. \tag{4.6}$$

The number of constraints of the above type (4.4) is linear in the number of vertices, while (4.5) and (4.6) are linear in the number of lifted edges. Hence, we add them into our initial constraint set during optimization.

Path inequalities. For lifted edge y'_{vw} it holds that if there is a flow in G going from v to w along a path P, then $y'_{vw} = 1$. This constraint can be expressed by the following set of inequalities:

$$\forall vw \in E'\ \forall P \in vw\text{-paths}(G): \quad y'_{vw} \geq \sum_{vj: j \in P_V} y_{vj} - \sum_{i \in P_V \setminus \{v,w\}} \sum_{k \notin P_V} y_{ik} \tag{4.7}$$

Here the first sum expresses the flow going from v to any vertex of path P. The second sum is the flow leaving path vertices P_V before reaching w. In other words, if the flow does not leave P_V, edge y'_{vw} must be active. Note that inequality (4.7) implicitly enforces

y'_{vw} to be active if any vw-path $\tilde{P}$ with $\tilde{P}_V \subset P_V$ is active and if there is no concurrent flow leaving any vertex in $P_V \setminus \tilde{P}_V$. Figure 4.2 illustrates the path inequalities.

For the multicut problem, there exist path inequalities that enforce path properties in an analogous way. While the lifted multicut path inequalities (3.4) would yield the same set of feasible integral points, the resulting polyhedral relaxation would be weaker. Proposition 4.1 states this fact. Note that this chapter uses a convention for zeros and ones that is opposite to the convention from Chapter 3. While label one in (LDP) denotes edges within components (paths), label one on an edge in (LMC) means that its endpoints belong to different components. Inequality (4.8) is the lifted multicut path inequality (3.4) rewritten in the opposite convention.

Proposition 4.1. *Path inequalities (4.7) define a strictly tighter relaxation of the lifted disjoint paths problem than the lifted multicut path inequalities*

$$\forall vw \in E' \; \forall P \in vw\text{-}paths(G): \quad y'_{vw} \geq \sum_{ij \in P_E} (y_{ij} - 1) + 1\,. \tag{4.8}$$

Proof. Let us define the following sets:

$$S_B = \{(y, y') \in [0,1]^E \times [0,1]^{E'} | (y, y') \text{ satisfy } (4.7)\}\,,$$
$$S_M = \{(y, y') \in [0,1]^E \times [0,1]^{E'} | (y, y') \text{ satisfy } (4.8)\}\,.$$

- Let us prove that $S_B \subset S_M$

  Let us rewrite the right-hand side of (4.7) for a path $P \in vw$-paths(G):

$$\begin{aligned} y'_{vw} &\geq \sum_{vj: j \in P_V} y_{vj} - \sum_{i \in P_V \setminus \{v,w\}} \sum_{k \notin P_V} y_{ik} = \sum_{vj: j \in P_V} y_{vj} - \sum_{i \in P_V \setminus \{v,w\}} (z_i - \sum_{j \in P_V} y_{ij}) = \\ &= \sum_{i \in P_V \setminus w} \sum_{j \in P_V} y_{ij} - \sum_{i \in P_V \setminus \{v,w\}} z_i \geq \sum_{ij \in P_E} y_{ij} - \sum_{i \in P_V \setminus \{v,w\}} 1 = \\ &= \sum_{ij \in P_E} (y_{ij} - 1) + 1\,. \end{aligned} \tag{4.9}$$

- Let us prove that $S_B \subsetneq S_M$

  We prove that there exists $(y, y') \in [0,1]^E \times [0,1]^{E'}$ such that (y, y') satisfies (4.8) and does not satisfy (4.7). An example is given in Figure 4.3. There are four possible paths from v to w. If we use Constraints (4.8), the highest lower bound on y'_{vw} is given by path $P = (vv_2, v_2v_4, v_4w)$ and it is as follows:

$$y'_{vw} \geq (0.5 - 1) + (0.5 - 1) + (1 - 1) + 1 = 0\,. \tag{4.10}$$

  Let us apply Constraint (4.7) using path $P = (vv_1, v_1v_2, v_2v_3, v_3v_4, v_4w)$. We obtain the following threshold on y'_{vw}

$$y'_{vw} \geq 0.5 + 0.5 - 0 - 0 = 1\,. \tag{4.11}$$

□

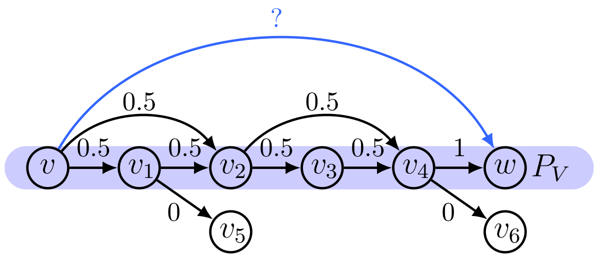


Figure 4.3: A failure case for the lifted multicut path inequality (4.8). The path inequality (4.7) gives the correct lower bound for lifted edge y'_{vw} in this case. Example for Proposition 4.1.

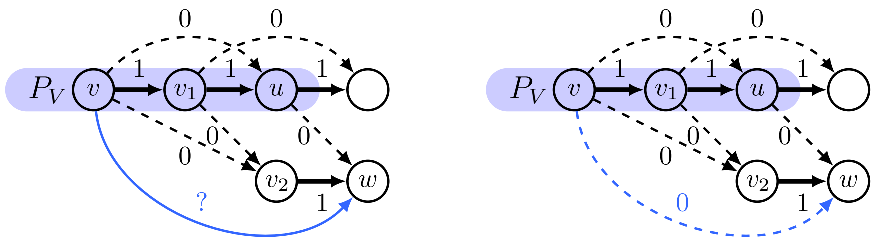


Figure 4.4: An illustration of the path-induced cut inequalities (4.12). The label of lifted edge vw (represented by the question mark in the left figure) is determined by the zero-labeled cut separating active path $P = (vv_1, v_1u)$ from w. In particular, $y'_{vw} \leq y_{vv_2} + y_{v_1v_2} + y_{uw} = 0$. Consequently, $y'_{vw} = 0$ (right figure).

Path-induced cut inequalities. The path-induced cut inequalities generalize the single node cut inequalities (4.5) and (4.6) by allowing cuts induced by paths.

Let a lifted edge $vw \in E'$, a node u from which w is reachable and a vu-path P be given. Consider the cut given by edges ik with $i \in P_V$ and $k \notin P_V$ but such that w is reachable from k. If the flow does not take any edge of this cut, then $y'_{vw} = 0$. Formally,

$$\forall vw \in E' \ \forall P \in vu\text{-paths}(G) \text{ s.t. } uw \in \mathcal{R}_G, u \neq w: \quad y'_{vw} \leq \sum_{i \in P_V} \sum_{\substack{k \notin P_V, \\ kw \in \mathcal{R}_G}} y_{ik}\,. \tag{4.12}$$

See Figure 4.4 for an illustration of the path-induced cut inequalities (4.12).

Lifted inequalities. The path inequalities (4.7) and the path-induced cut inequalities (4.12) only consider base edges on their right-hand sides. We can generalize both (4.7) and (4.12) by including lifted edges in the paths as well. Conceptually, using lifted edges allows representing all possible paths between their endpoints, which enables to formulate tighter inequalities, see Propositions 4.2 and 4.3.

To that end consider the multigraph $G \cup G' := (V, E \cup E')$. For any edge $ij \in E \cap E'$ we always distinguish whether $ij \in E$ or $ij \in E'$. For $P \in vw$-paths$(G \cup G')$, we denote by P_E and $P_{E'}$ edges of the path P in E and E' respectively. We require $P_E \cap P_{E'} = \emptyset$.

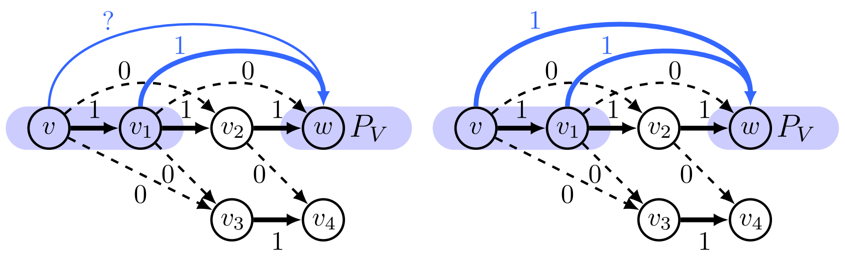


Figure 4.5: An illustration of the lifted path inequalities (4.13). Given the active path $P = (vv_1, v_1w)$, where $P_E = \{vv_1\}$ and $P_{E'} = \{v_1w\}$, the label of lifted edge vw (labeled with question mark in the left figure) is determined by inequality $y'_{vw} \geq y_{vv_1} - y_{v_1v_2} - y_{v_1v_3} + y'_{v_1w} - y_{v_1w} = 1 - 1 - 0 + 1 - 0 = 1$. Therefore, $y'_{vw} = 1$ (right figure).

Lifted path inequalities. We generalize the path inequalities (4.7). Now the vw-path P may contain both edges in E and E'. Whenever a lifted edge y'_{ij} in the third sum in (4.13) is one, two cases can occur: (i) Flow goes out of P (uses vertices not in P_V) but reenters it again later. Then a base edge variable y_{ik} will be one in the second sum in (4.13) and the values of y'_{ij} and y_{ik} cancel out. (ii) A base edge $ij \in E \cap E'$ parallel to the lifted edge is active. Then the variable y_{ij} in the fourth sum in (4.13) cancels out y'_{ij}. The lifted path inequalities become

$$\forall vw \in E' \; \forall P \in vw\text{-paths}(G \cup G') :$$
$$y'_{vw} \geq \sum_{j \in P_V} y_{vj} - \sum_{i \in P_V \setminus \{v,w\}} \sum_{k \notin P_V} y_{ik} + \sum_{ij \in P_{E'}} y'_{ij} - \sum_{ij \in P_{E'} \cap E} y_{ij} \,. \tag{4.13}$$

Whenever the path in (4.13) consists only of base edges P_E, the resulting inequality becomes a path inequality (4.7). Figure 4.5 illustrates the lifted path inequalities (4.13).

Proposition 4.2. *The lifted path inequalities* (4.13) *provide a strictly better relaxation than the path inequalities* (4.7).

Proof. Let us define the following sets

$$S_B = \{(y, y') \in [0,1]^E \times [0,1]^{E'} | (y, y') \text{ satisfy } (4.7)\} \,,$$
$$S_L = \{(y, y') \in [0,1]^E \times [0,1]^{E'} | (y, y') \text{ satisfy } (4.13)\} \,.$$

- Let us prove that $S_L \subset S_B$:
  Note that every path $P \in vw\text{-paths}(G)$ belongs to the set of $vw\text{-paths}(G \cup G')$ too. It just holds that $P_{E'} = \emptyset$. Let us rewrite the right-hand side of the inequality from (4.13) for such $P \in vw\text{-path}(G \cup G')$ where $P_{E'} = \emptyset$.

$$y'_{vw} \geq \sum_{vj : j \in P_V} y_{vj} - \sum_{i \in P_V \setminus \{v,w\}} \sum_{k \notin P_V} y_{ik} + \sum_{ij \in P_{E'}} y'_{ij} - \sum_{ij \in P_{E'} \cap E} y_{ij} =$$
$$= \sum_{vj : j \in P_V} y_{vj} - \sum_{i \in P_V \setminus \{v,w\}} \sum_{k \notin P_V} y_{ik} \,. \tag{4.14}$$

  Which is exactly the right-hand side of (4.7). Therefore, any pair of real vectors $(y, y') \in [0,1]^E \times [0,1]^{E'}$ that satisfies (4.13) must satisfy (4.7) as well.

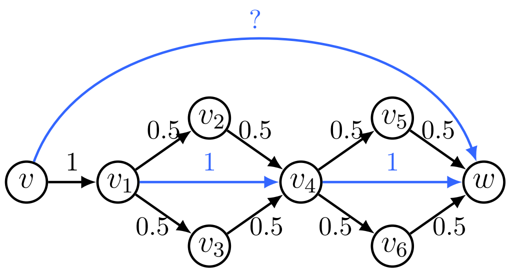


Figure 4.6: Exemplary case where the path inequalities (4.7) give a trivial lower bound on lifted edge y'_{vw}. The lifted path inequality (4.13) gives the correct lower bound. Example for Proposition 4.2.

- Let us prove that $S_L \subsetneq S_B$:
  We prove that there exists $(y, y') \in [0,1]^E \times [0,1]^{E'}$ such that (y, y') satisfies (4.7) and does not satisfy (4.13). See the graph in Figure 4.6. There are four possible paths from v to w in G. If we use Constraints (4.7), all the paths give us the same lower bound on y'_{vw}

$$y'_{vw} \geq 1 - 0.5 - 0.5 = 0 \ . \tag{4.15}$$

If we use Constraints (4.13) with path $P = (vv_1, v_1v_4, v_4w)$ where $P_{E'} = \{v_1v_4, v_4w\}$, we obtain

$$y'_{vw} \geq 1 - 0.5 - 0.5 - 0.5 - 0.5 + 1 + 1 = 1 \ . \tag{4.16}$$

□

Lifted path-induced cut inequalities. We generalize the path-induced cut inequalities (4.12). Let a lifted edge $vw \in E'$ and a vu-path P in $G \cup G'$ be given. In contrast to the basic version (4.12), a lifted edge $ij \in P_{E'}$ can be taken. This can occur in two cases: Either the flow leaves P_V via a base edge ik, $k \notin P_V$ or a base edge $ij \in E \cap E'$ parallel to the lifted edge is taken. Both cases are accounted for by terms in the first and the third sum in (4.17) below.

$$\begin{aligned} &\forall vw \in E' \ \forall P \in vu\text{-paths}(G \cup G') \text{ s.t. } \ uw \in \mathcal{R}_G \wedge u \neq w : \\ &y'_{vw} \leq \sum_{i \in P_V} \sum_{\substack{k \notin P_V, \\ kw \in \mathcal{R}_G}} y_{ik} - \sum_{ij \in P_{E'}} y'_{ij} + \sum_{ij \in P_{E'} \cap E} y_{ij} \end{aligned} \tag{4.17}$$

An illustration of the lifted path-induced cut inequalities (4.17) is in Figure 4.7.

Assume that the last node u of path P is connected via a lifted edge with w. Then

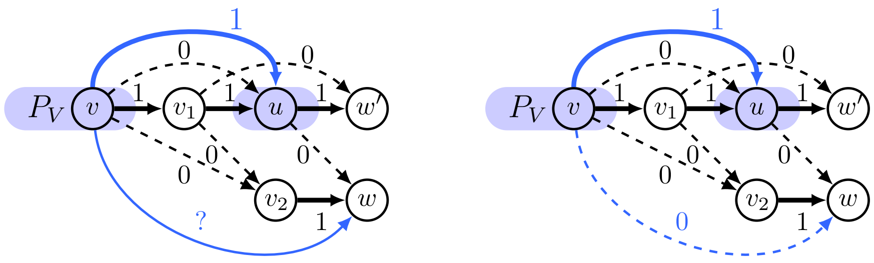


Figure 4.7: An illustration of lifted path-induced cut inequalities (4.17). Given path $P = (vu)$ where $P_{E'} = \{vu\}$, the label of lifted edge vw (represented by the question mark in the left figure) is determined by the following constraint $y'_{vw} \leq y_{vv_1} + y_{vv_2} + y_{uw} - y'_{vu} + y_{vu} = 1+0+0-1+0 = 0$. Therefore, $y'_{vw} = 0$ (right figure).

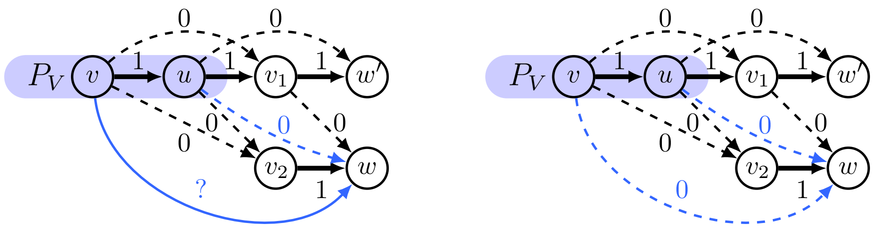


Figure 4.8: An illustration of lifted path-induced cut inequalities (4.18). Given the path $P = (vu)$ where $P_E = \{vu\}$ and the zero-labeled lifted edge uw, the label of lifted edge vw (represented by the question mark in the left figure) is determined by constraint $y'_{vw} \leq y_{vv_1} + y_{vv_2} + y'_{uw} = 0$. Consequently, $y'_{vw} = 0$ (right figure).

we can strengthen (4.17) by replacing the sum of base edges outgoing from u by y'_{uw}.

$$\forall vw \in E' \; \forall P \in vu\text{-paths}\,(G \cup G') \;\; \text{s.t. } uw \in E' :$$
$$y'_{vw} \leq \sum_{i \in P_V \setminus u} \sum_{\substack{k \notin P_V, \\ kw \in \mathcal{R}_G}} y_{ik} - \sum_{ij \in P_{E'}} y'_{ij} + \sum_{ij \in P_{E'} \cap E} y_{ij} + y'_{uw} \tag{4.18}$$

An illustration of the lifted path-induced cut inequalities (4.18) is in Figure 4.8.

Proposition 4.3. *The lifted path-induced cut inequalities* (4.17) *define a strictly tighter relaxation than the path-induced cut inequalities* (4.12).

Furthermore, the lifted path-induced cut inequalities (4.17) *and* (4.18) *define a strictly better relaxation than* (4.17) *alone.*

Proof. Let us define the following sets

$$S_B = \{(y, y') \in [0,1]^E \times [0,1]^{E'} | (y, y') \text{ satisfy } (4.12)\},$$
$$S_{L1} = \{(y, y') \in [0,1]^E \times [0,1]^{E'} | (y, y') \text{ satisfy } (4.17)\},$$
$$S_{L2} = \{(y, y') \in [0,1]^E \times [0,1]^{E'} | (y, y') \text{ satisfy } (4.18)\}.$$

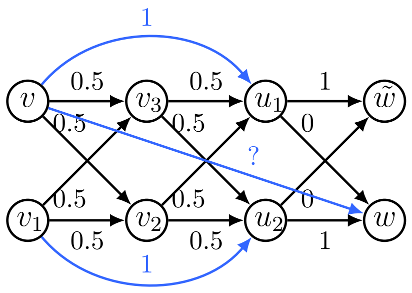


Figure 4.9: An Exemplary case where the path-induced cut inequalities (4.12) fail to give non-trivial upper bounds for lifted edge y'_{vw}. The lifted path-induced cut inequalities (4.17) give the correct upper bound in this case. Example for Proposition 4.3.

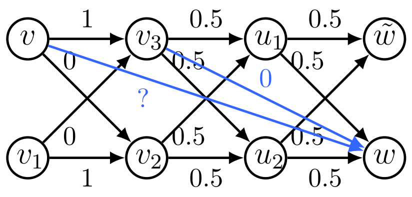


Figure 4.10: An Exemplary failure case for the lifted path-induced cut inequalities (4.17). The lifted path-induced cut inequalities (4.18) give the correct upper bound for lifted edge y'_{vw}. Example for Proposition 4.3.

- First, we prove $S_{L1} \subset S_B$:
  We use the same argument as in the proof of Proposition 4.2. Every path $P \in vw\text{-paths}(G)$ belongs to the set of $vw\text{-paths}(G \cup G')$ and it holds that $P_{E'} = \emptyset$. Let us rewrite the right-hand side of the inequality from (4.17) for such $P \in vw\text{-path}(G \cup G')$ where $P_{E'} = \emptyset$.

$$y'_{vw} \leq \sum_{i \in P_V} \sum_{\substack{k \notin P_V \\ kw \in \mathcal{R}_G}} y_{ik} - \sum_{ij \in P_{E'}} y'_{ij} + \sum_{ij \in P_{E'} \cap E} y_{ij} = \sum_{i \in P_V} \sum_{\substack{k \notin P_V \\ kw \in \mathcal{R}_G}} y_{ik} \,. \tag{4.19}$$

  Which is exactly the right-hand side of (4.12). Therefore, any pair of real vectors $(y, y') \in [0,1]^E \times [0,1]^{E'}$ that satisfies (4.17) must satisfy (4.12).

- Let us prove $S_{L1} \subsetneq S_B$:
  We prove that there exists $(y, y') \in [0,1]^E \times [0,1]^{E'}$ such that (y, y') satisfies (4.12) and does not satisfy (4.17).

  See the example in Figure 4.9. There are four possible paths in G from v to either u_1 or u_2. They are $P_1 = (vv_3, v_3u_1)$, $P_2 = (vv_2, v_2u_1)$, $P_3 = (vv_3, v_3u_2)$, $P_4 = (vv_2, v_2u_2)$. Using (4.17), all of them give us the same threshold on y'_{vw}:

$$y'_{vw} \leq 0.5 + 0.5 + 0 = 1 \,. \tag{4.20}$$

If we use Constraint (4.17) with path $P = (vu_1)$, we obtain the following threshold:

$$y'_{vw} \leq 0.5 + 0.5 + 0 - 1 = 0\,. \tag{4.21}$$

- Let us prove that $S_{L1} \cap S_{L2} \subsetneq S_{L1}$
  It holds trivially that $S_{L1} \cap S_{L2} \subset S_{L1}$. Let us prove that there exists $(y, y') \in [0,1]^E \times [0,1]^{E'}$ such that $(y, y') \in S_{L1}$ and $(y, y') \notin S_{L1} \cap S_{L2}$.

  See the example graph in Figure 4.10. Similarly as in Figure 4.9, there are four possible paths from v to either u_1 or u_2 in G. There are no active lifted edges that would enable us to obtain a better upper bound on y'_{vw} using (4.17) than the following:

$$y'_{vw} \leq 1\,. \tag{4.22}$$

  However, if we use Constraints (4.18) with path $P = (vv_3)$ and $y'_{v_3w} = 0$, we obtain

$$y'_{vw} \leq 0\,. \tag{4.23}$$

□

Symmetric form of cut inequalities. Inequalities (4.6) provide a symmetric counterpart to inequalities (4.5). We can also formulate symmetric counterparts to inequalities (4.12), (4.17) and (4.18) by swapping the role of v and w. All constraints (4.12), (4.17) and (4.18) concentrate on paths originating in v. The symmetric inequalities are obtained by studying all paths ending in w.

Relations analogous to those described in Proposition 4.3 hold for the symmetric counterparts as well. The symmetric inequalities also strengthen the relaxation strictly.

Inequalities symmetric to (4.12):

$$\begin{gathered} \forall vw \in E'\ \forall P \in uw\text{-paths}(G) \text{ s.t. } \ vu \in \mathcal{R}_G \wedge u \neq v : \\ y'_{vw} \leq \sum_{i \in P_V} \sum_{\substack{k \notin P_V, \\ vk \in \mathcal{R}_G}} y_{ki}\,. \end{gathered} \tag{4.24}$$

Inequalities symmetric to (4.17)

$$\begin{gathered} \forall vw \in E'\ \forall P \in uw\text{-paths}(G \cup G') \text{ s.t. } \ vu \in \mathcal{R}_G \wedge u \neq v : \\ y'_{vw} \leq \sum_{i \in P_V} \sum_{\substack{k \notin P_V, \\ vk \in \mathcal{R}_G}} y_{ki} - \sum_{ij \in P_{E'}} y'_{ij} + \sum_{ij \in P_{E'} \cap E} y_{ij}\,. \end{gathered} \tag{4.25}$$

Inequalities symmetric to (4.18)

$$\begin{gathered} \forall vw \in E'\ \forall P \in uw\text{-paths}\ (G \cup G') \ \text{ s.t. } vu \in E' : \\ y'_{vw} \leq \sum_{i \in P_V \setminus u} \sum_{\substack{k \notin P_V, \\ vk \in \mathcal{R}_G}} y_{ki} - \sum_{ij \in P_{E'}} y'_{ij} + \sum_{ij \in P_{E'} \cap E} y_{ij} + y'_{vu}\,. \end{gathered} \tag{4.26}$$

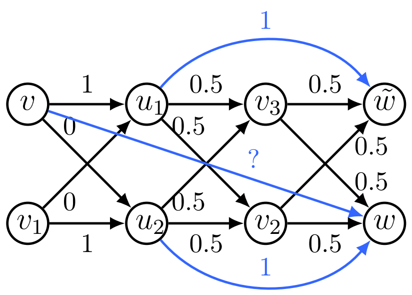


Figure 4.11: The best upper bound on y'_{vw} is provided by inequalities (4.25). Example for Proposition 4.4 and Proposition 4.5.

Proposition 4.4. *The lifted path-induced cut inequalities (4.25) define a strictly tighter relaxation than the path-induced cut inequalities (4.24).*
The lifted path-induced cut inequalities (4.25) and (4.26) define a strictly better relaxation than (4.25) alone.

Proof. Analogical to the proof of Proposition 4.3. See Figure 4.11 for example analogical to the one in Figure 4.9 and Figure 4.12 for example analogical to the one in Figure 4.10.
□

Proposition 4.5. *The following claims about the path-induced cut inequalities hold.*

1. *The path-induced cut inequalities (4.12) together with their symmetric counterpart (4.24) define a strictly tighter relaxation than inequalities (4.12) alone.*

2. *The path-induced cut inequalities (4.17) together with their symmetric counterpart (4.25) define a strictly tighter relaxation than inequalities (4.17) alone.*

3. *Using path-induced cut inequalities (4.26) together with (4.17), (4.18), and (4.25) strictly improves the relaxation.*

Proof. 1. See the example in Figure 4.13.
Upper bound on y'_{vw} by (4.12): $y'_{vw} \leq 0.5 + 0.5 = 1$.
Upper bound on y'_{vw} by (4.24): $y'_{vw} \leq 0$.

2. See the example in Figure 4.11.
Upper bound on y'_{vw} by (4.17): $y'_{vw} \leq 0.5 + 0.5 = 1$.
Upper bound on y'_{vw} by (4.25) using path $P = (u_2 w)$: $y'_{vw} \leq 0 + 0.5 + 0.5 - 1 = 0$.

3. See the example in Figure 4.12.
Upper bounds on y'_{vw} by (4.17), (4.18), (4.25): $y'_{vw} \leq 1$. Upper bound on y'_{vw} by (4.26) using path $P = (uw)$ and $y'_{vu} = 0$: $y'_{vw} \leq 0$.
□

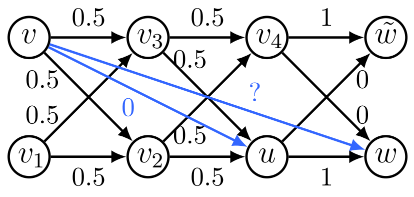


Figure 4.12: The best upper bound on y'_{vw} is provided by inequalities (4.26). Example for Proposition 4.4 and Proposition 4.5.

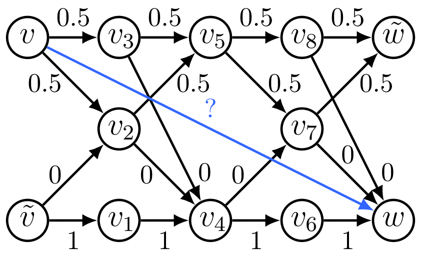


Figure 4.13: The best upper bound on y'_{vw} is provided by inequalities (4.24). Example for Proposition 4.5.

Other valid inequalities. Basic flow constraints (4.4) together with the advanced constraints on lifted edges (4.5)-(4.18) are sufficient for defining the set of feasible solutions of the lifted disjoint paths problem (LDP). Moreover, they define an efficient LP relaxation (Section 4.4) and enable efficient separation procedures (Section 4.5). Below, we present lifted flow inequalities specific to the lifted disjoint paths problem applied to MOT that help to improve the speed of our ILP solver. The inequalities depend on the fact that every node can be connected to maximally one node in each time frame. Therefore the number of lifted edges originating (or ending) in a given point and ending (resp. originating) in a specific time frame is at most one.

$$\forall k, l \in \{1, \ldots, T\} : k > l,\ \forall v \in V_l : \sum_{vu \in E' : u \in V_k} y'_{vu} \leq z_v\,, \tag{4.27}$$

$$\forall k, l \in \{1, \ldots, T\} : k < l,\ \forall w \in V_l : \sum_{uw \in E' : u \in V_k} y'_{uw} \leq z_w\,. \tag{4.28}$$

The number of constraints (4.27) and (4.28) is linear in the number of vertices. Therefore, we add them to our initial constraint set. This enables to reduce the search space for the branch and cut method in the early solver stages when only a few constraints of type (4.7)-(4.18) have been added.

Algorithm Separate-lifted-path-inequalities (4.13)

```
Define $E^1 = \{e \in E : y_e = 1\}$, $G^1 = (V, E^1)$
for all $P^1 \in st$-paths $(G^1)$ do
    for all $y'_{vw} = 0 : v \in P^1_V \wedge w \in P^1_V$ do
        $P :=$ Extract-path$(P^1, v, w)$
        Add constr. (4.13) for $y'_{vw}$ with $P$.
    end for
end for
```

4.5 SEPARATION

We solve the lifted disjoint paths problem (LDP) with the state-of-the-art integer linear program solver Gurobi (Gurobi Optimization, 2019). Since there are exponentially many constraints of the form (4.7), (4.12), (4.13), (4.17) and (4.18), we do not add them initially. Instead, we start with constraints (4.4), (4.5), (4.6) and eventually (4.27) and (4.28). Subsequently, we find the optimal integer solution. In the separation procedures described below, we check if any of the advanced constraints are violated and add those that are to the active constraint set. We resolve the tightened problem and iterate until we have found a feasible solution to the overall problem (LDP).

Section (4.7) describes a two-step procedure that we use for the processing of the whole sequences. We add inequalities (4.27) and (4.28) only in the first step of the procedure.

Algorithms Separate-lifted-path-inequalities and Separate-lifted-path-induced-cut-inequalities describe the separation procedures for adding lifted path constraints (4.13), and lifted path-induced cut constraints (4.17) and (4.18). Since path constraints (4.7) and path-induced cut inequalities (4.12) are special cases of those above, they are also accounted for.

Separation for path inequalities. Algorithm Separate-lifted-path-inequalities iterates over all active st-paths. For every path P^1, labels of all lifted edges connecting two vertices in P^1_V are inspected. If the lifted edge variable is zero, Algorithm Separate-lifted-path-inequalities will extract a path in $G \cup G'$ connecting the endpoints and add the resulting lifted path inequality (4.13) to the active constraint set.

Separation for path-induced cut inequalities. Algorithm Separate-lifted-path-induced-cut-inequalities iterates over all active st-paths. For every path P^1, lifted edges that start in P^1_V but do not end in P^1_V are inspected. If their label is one, Algorithm Separate-lifted-path-induced-cut-inequalities will extract a subpath of P^1 for either (4.18) or (4.17) and add the respective inequality to the active constraint set.

The complexity of separation. Both Algorithms Separate-lifted-path-inequalities and Separate-lifted-path-induced-cut-inequalities can be implemented efficiently such that they are linear in $|E^1|$ (i.e. in the number of active edges of graph G). In our implementation, we traverse all active st-paths from the end to the beginning and directly store correctly

Algorithm Separate-lifted-path-induced-cut-inequalities (4.17), (4.18)

```
Define $E^1 = \{e \in E : y_e = 1\}$, $G^1 = (V, E^1)$
for all $P^1 \in st$-paths $(G^1)$ do
    for all $y'_{vw} = 1 : v \in P^1_V \wedge w \notin P^1_V$ do
        if $\exists u \in P^1_V : y'_{uw} = 0 \wedge vu \in \mathcal{R}_G$ then
            $P :=$ Extract-path$(P^1, v, u)$
            Add constr. (4.18) for $y'_{vw}$ with $P$.
        else
            $u :=$ last vertex of $P^1$ such that $uw \in \mathcal{R}_G$
            $P :=$ Extract-path$(P_1, v, u)$
            Add constr. (4.17) for $y'_{vw}$ with $P$.
        end if
    end for
end for
```

Algorithm Extract-path (P^1, v, w)

```
$P' := vw$-subpath of $P^1$, $P := \emptyset$
for $j \in P'_V$ from end of path to beginning do
    if $\exists$ edge $ij \in E'$, $i \in P'_V, y'_{ij} = 1$ then
        Add $ij$ to $P_{E'}$, skip to node $i \in P'_V$
    else
        Add $ij$ from $P'$ to $P_E$
    end if
end for
output $P = P_E \cup P_{E'}$
```

labeled lifted edges that originate on the already processed subpaths. These lifted edges can be used later as edges in $P_{E'}$ in (4.13)-(4.18) or as $y'_{uw} = 0$ in (4.18).

4.6 COMPLEXITY

Below, we show that the lifted disjoint paths problem (LDP) is NP-hard. The following theorems state that even its restricted versions using only negative or only positive lifted edges are NP-hard. The proofs use reductions from two known NP-complete problems. Theorem 4.1 is proven by reduction from the integer multicommodity flow (Even *et al.*, 1975) and Theorem 4.2 by reduction from 3-SAT (Cook, 1971).

We define $Y_{GG'}$ to be the set of all $(y, y') \in \{0,1\}^E \times \{0,1\}^{E'}$ such that (y, y') are feasible solutions of the lifted disjoint paths problem (LDP).

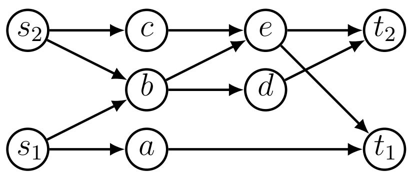


Figure 4.14: Integer multicommodity flow network transformation (Lemma 4.1): Original graph.

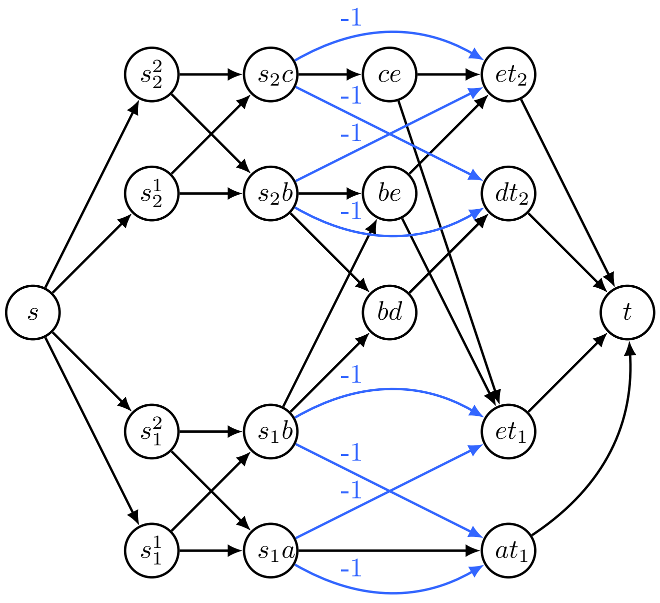


Figure 4.15: Integer multicommodity flow network transformation (Lemma 4.1). Transformed graph from Figure 4.14 for flow requirements $R_1 = 2, R_2 = 2$. Edges without labels have cost 0.

4.6.1 INTEGER MULTICOMMODITY FLOW.

The integer multicommodity flow problem is defined on a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with edge capacities $c \in \mathbb{N}^{\mathcal{E}}$ and source/sink pairs $s_i t_i$ and edge flows $f_i \in \mathbb{N}^{\mathcal{E}}$ and requirements R_i, $i = 1, \ldots, k$.

The aim is to send k flows from their sources to their sinks such that the flows obey the edge capacities. Formally,

$$\sum_{i=1}^{k} f_e^i \leq c_e \qquad \forall e \in E \tag{4.29}$$

$$\sum_{u:uv \in E} f_{uv}^i = \sum_{w:vw \in E} f_{vw}^i \qquad \forall i \in [k]\ \forall v \notin \{s_i, t_i\} \tag{4.30}$$

$$\sum_{v:s_i v \in E} f_{s_i v}^i \geq R_i \qquad \forall i \in [k] \tag{4.31}$$

where $[k]$ denotes the set $\{1, \ldots, k\}$. Even *et al.* (1975) have shown that the integer multicommodity flow problem is NP-complete also in the case of unit capacity edges and two source-sink pairs. Below we detail a construction that gives us a correspondence between edge-disjoint paths in $\mathcal{G}$ and node-disjoint paths in the transformed graph G. This construction is similar to transforming a graph into its line graph. The lifted edges in the transformed graph will count how many units of flow go from sources to sinks.

Lemma 4.1. *There exists a polynomial transformation from any graph $\mathcal{G}$ with source/sink pairs s_i, t_i, $i = 1, \ldots, k$ with requirements R_i to a pair of graphs G and G' with edge costs c and c' respectively such that there exists a feasible integer multicommodity flow in $\mathcal{G}$ with unit edge capacities if and only if the lifted disjoint paths problem for G, G' has objective $\min_{(y,y') \in Y_{GG'}} \langle c, y\rangle + \langle c', y'\rangle \leq -\sum_{i=1}^{k} R_i$.*

Proof. Without loss of generality, we consider these feasible flow sets $f_1, \ldots, f_k$ where it holds $\forall i \in [k] : \sum_{s_i v \in E} f^i_{s_i v} = R_i$. Note that if the flow of commodity i is higher than its requirement R_i, we can reduce it to R_i by removing the flow across one or more $s_i t_i$-paths in $\mathcal{G}$ without violating other constraints.
We first detail the graph transformation (see Figures 4.14 and 4.15).

- For all edges $ij \in \mathcal{E}$ add a vertex v_{ij} to V.
- For each pair of vertices $v_{ij}, v_{jk} \in V$ add an edge (v_{ij}, v_{jk}) to E.
- Add vertices s and t to V.
- Add to V vertices $s_i^1, s_i^2, \ldots, s_i^{R_i}$ representing requirements of each commodity i.
- For each vertex s_i^r add an edge (s, s_i^r) to E.
- For each pair of vertices $s_i^r, v_{s_i j}$ add edge $(s_i^r, v_{s_i j})$ to E.
- For all $v_{kt_i} \in V$ (representing an edge from k to t_i in $\mathcal{G}$) add an edge (v_{kt_i}, t) to E.
- For all pairs of vertices $v_{s_i j}$ $v_{kt_i} \in V$ add an edge $(v_{s_i j}, v_{kt_i})$ to E'. That is, the lifted edges connect all vertices representing edges from s_i in $\mathcal{G}$ with vertices representing the edges to t_i in $\mathcal{G}$.
- Cost function on base edges $\forall e \in E : c_e = 0$.
- Cost function on lifted edges $\forall e' \in E' : c'_{e'} = -1$.

An illustration of this construction can be seen in Figures 4.14 and 4.15. Note that the construction of $\mathcal{G}$ in Even *et al.* (1975) allows $s_i = s_j$ for $i \neq j$. In this case, we still construct separate vertices for their incident edges in G.

Every path $\mathcal{P} = (s_i k_1, k_1 k_2, \ldots, k_n t_i)$ in $\mathcal{G}$ can be assigned to an st-path P in G where

$$P = (s s_i^r, s_i^r v_{s_i k_1}, v_{s_i k_1} v_{k_1 k_2}, \ldots, v_{k_n t_i} t)$$

and where $r \in [R_i]$ can be chosen arbitrarily and vice versa. Note that such a path P saturates exactly one lifted edge $(v_{s_i k_1}, v_{k_n t_i})$. Moreover, every feasible set of flow functions $f^1, \ldots, f^k$ satisfying for all $i \in [k] : \sum_{s_i v \in E} f^i_{s_i v} = R_i$ defines a set of edge-disjoint paths from $s_1, \ldots, s_k$ to $t_1, \ldots, t_k$ in $\mathcal{G}$. This set corresponds to a set of $\sum_{i=1}^{k} R_i$ st-paths in G whose edges and vertices are disjoint and where every path saturates exactly one lifted edge $v_{s_i j} v_{kt_i}$. Every lifted edge contributes with -1 to the total cost. So, this set

of disjoint st-paths has the total cost $-\sum_{i=1}^{k} R_i$.
Reversely, let us have a set of vertex-disjoint st-paths in G of size $\sum_{i=1}^{k} R_i$ where every path contains some $v_{s_ij}v_{kt_i}$-path as its subpath and therefore its cost is $-\sum_{i=1}^{k} R_i$. This set defines uniquely a set of feasible flow functions $f_1, \ldots, f_k$.
So, there exist feasible functions $f_1, \ldots, f_k$ satisfying $f_i = R_i$ for all $i \in [k]$ iff $\min\limits_{(y,y') \in Y_{GG'}} \gamma(y, y') \leq -\sum_{i=1}^{k} R_i$.

□

Modifications of the proof. As we discuss in Section 2.3.2, there are polynomial time reductions between the directed node-disjoint paths problem and directed edge-disjoint paths problem. Therefore, the transformation in the proof of Lemma 4.1 can be done for the node-disjoint paths problem directly. In that case, it is just necessary to create R_i duplicates of each source node s_i resp. sink node t_i. Moreover, it is sufficient to create one lifted edge between each pair of source-sink duplicates (s_i^r, t_i^r) only. This way, the number of lifted edges is equal to the total flow requirement. That is, $|E'| = \sum_{i=1}^{k} R_i = |D|$ where $H = (T, D)$ is the demand graph.

Theorem 4.1. *Lifted disjoint paths problem* (LDP) *with negative lifted edges only is NP-hard.*

Proof. The NP-complete integer multicommodity flow problem with unit edge capacities can be reduced in polynomial time to the lifted disjoint paths problem (LDP) with negative lifted edges only. The transformation is described in Lemma 4.1.

□

4.6.2 3-SAT PROBLEM

The boolean satisfiability problem (SAT) is a classical NP-complete problem (Cook, 1971). A transformation from its NP-complete special version 3-SAT is commonly used for proving that a problem is NP-hard or NP-complete. Note that the used transformation is analogical to the one used for proving Theorem 3.1.

Theorem 4.2. *Lifted disjoint paths problem* (LDP) *with positive lifted edges only is NP-hard.*

Proof. Below, we detail a transformation from 3-SAT to the lifted disjoint paths problem with positive lifted edges only. For the transformation, it holds that a 3-SAT formula consisting of k clauses has a true assignment iff $\min\limits_{(y,y') \in Y_{GG'}} \langle c, y \rangle + \langle c', y' \rangle \leq -(k-1)$.

Let a 3-SAT problem containing k ordered clauses $C_1 \ldots C_k$ be given. Each clause C_i consists of a conjunction of literals, which is either a variable a or its complement $\overline{a}$. We construct graphs $G = (V, E)$ and $G' = (V', E')$ as stated below. See also Figure 4.16.

- The graph G has k layers. Every layer corresponds to one clause. Each layer contains 3 vertices labeled with the literals in the corresponding clause. Specifically, for a variable a in clause C_i we associate node v_{ia}, analogically for a complemented variable $\overline{b}$ in clause C_i we associate node $v_{i\overline{b}}$.

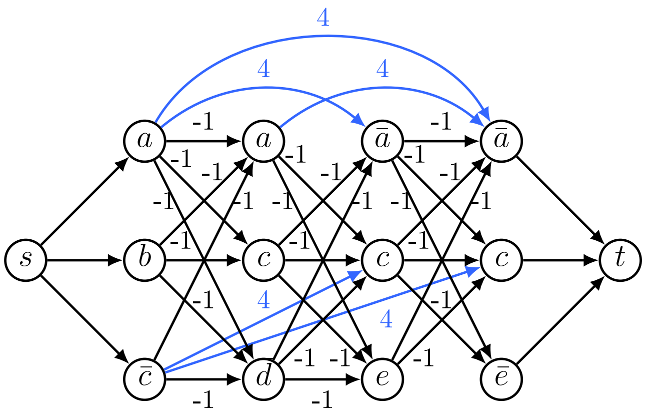


Figure 4.16: Reduction to lifted disjoint paths problem for 3-SAT formula $(a \vee b \vee \bar{c}) \wedge (a \vee c \vee \bar{d}) \wedge (\bar{a} \vee c \vee e) \wedge (\bar{a} \vee c \vee \bar{e})$, see the proof Theorem 4.2.

- For every pair of vertices $v_{il_1} \in V$ and $v_{i+1l_2} \in V$ where $l_1 \neq \bar{l_2}$ add an edge (v_{il_1}, v_{i+1l_2}) to E and set $c_{(v_{il_1}, v_{i+1l_2})} = -1$.

- For every variable a and every pair of vertices $v_{ia}, v_{j\bar{a}} \in V$ where $j > i+1$ add an edge $(v_{ia}, v_{j\bar{a}})$ to E' and set $c'_{(v_{ia}, v_{j\bar{a}})} = k$. Do so analogically for every pair of variables $v_{i\bar{a}}$ and v_{ja}.

- Add an edge from s to all vertices corresponding to the first clause. And an edge to t from all vertices corresponding to the last clause.

Every path $P \in st$-paths(G) that has cost $-(k-1)$ saturates vertices labeled by non-contradicting literals. We can obtain a 3-SAT solution from P as follows. If $v_{ia} \in P_V$, set variable $a := true$. If $v_{j\bar{b}} \in P$, set variable $b := false$. Variables not contained as labels of vertices in P_V can have arbitrary values.
Similarly, every solution of the 3-SAT problem defines at least one path $P \in st$-paths(G) that has cost $-(k-1)$.

□

4.7 EXPERIMENTS

We conduct several experiments on MOT datasets showing the merit of using lifted disjoint paths for the tracking problem. Below, we present our complete tracking method called LifT. We describe our problem construction, cost learning for the base and lifted edges, preprocessing and post-processing steps, and report resulting performance. Further details to the experiments are provided in the Appendix of Horňáková *et al.* (2020).

4.7.1 GRAPH CONSTRUCTION.

Two-step procedure. Due to the computational complexity of the problem, we cannot solve entire video sequences straightforwardly. In order to make the problem tractable, we apply the following two-step procedure. In the first step, the solver is applied on graphs over person detections but only for small time intervals consisting of a few dozen video frames. The tracks resulting from the first step are used for extracting tracklets. In the second step, the solver is applied on newly created graphs G and G' where vertices correspond to the obtained tracklets. Edges and edge costs between tracklets are obtained by aggregating original edges resp. edge costs between person detections. The tracks resulting from the second step may be suboptimal with respect to the original objective function defined over person detections. Therefore, we identify points where splitting a track leads to an improvement of the original objective value and extract new tracklets from the divided tracks. Multiple iterations of the second step are performed until no improving split points are found in the output tracks. This two-step procedure improves the objective w.r.t. the original objective (LDP) in every iteration. Since there are only finitely many trackings, the procedure terminates finitely. In practice, only a few iterations are necessary.

Graph sparsification. For our experiments, we use edges between detections up to 2sec temporal distance. These long-range edges cause high computational complexity for the first step. In order to reduce it, we apply sparsification on both base and lifted graphs. For the base edges, we select for every $v \in V \setminus \{s, t\}$ its K nearest (lowest-cost) neighbors from every subsequent time frame within an allowed time gap. Lifted edges with costs close to zero are not included, since they are not discriminative. Lifted edges connecting detections with high time gaps are included more sparsely than lifted edges having lower time gaps. We use dense graphs in the second step.

Costs. Initially, in the first step, we set the cost of all vertices $v \in V$ to $d_v = 0$. For the second step, where V represents tracklets, d_v is set to the cost of outputting tracklet v as the final trajectory. Specifically, d_v is the sum of costs of base edges between consecutive detections in the tracklet and the cost of lifted edges between all pairs of detections contained in the tracklet. The cost of a base edge between two tracklets is given by the cost of the original base edge connecting the last detection in the first tracklet with the first detection in the subsequent tracklet. The cost of a lifted edge between two tracklets is obtained by summing up the costs of original lifted edges between detections contained in the tracklets. This ensures that the cost of each tracklet solution corresponds to the cost of the solution of the original problem. We set the costs of all edges from the source node s and to the sink node t to zero. Setting of detection costs and in/out costs to zero reduces the number of hyperparameters that usually need to be incorporated by other methods. Moreover, our method does not include temporal decay of edge costs since the formulation directly prefers short-range base edges over the long-range ones.

4.7.2 PREPROCESSING AND POST-PROCESSING

As is common for tracking by detection, we perform pre- and post-processing to compensate for detector inaccuracies.

Input filtering. Given a set of input detections derived from a detector, we follow the approach of Bergmann *et al.* (2019), a leading tracker for the MOT challenge, to reject false positive detections and to correct misaligned ones. For this, each input detection is sent through the regression and classification part of their detector. In more detail, all tracking parts involved in the tracker Tracktor (Bergmann *et al.*, 2019) are deactivated, such that it only reshapes and eventually rejects input detections, without assigning labels to them. Input detections are rejected if Tracktor's detector outputs a confidence score $\sigma_{\text{active}} \leq 0.5$.

Tracktor also applies a non-maxima-suppression on the reshaped input detections, where we use the threshold $\lambda_{\text{new}} = 0.6$.

Interpolation and extrapolation. Even if all input detections have been assigned to the correct identities by our ILP solver, there might still be missing detections in case that a person has not been detected in some frames. We recover missing detections within the time range of a trajectory, which we denote as interpolation. Further, we extend a trajectory in forward and backward directions, which we denote as extrapolation. To this end, we follow Bergmann *et al.* (2019) and apply their object detector to recover missing positions based on the visual information at the last known position. Finally, for sequences filmed from a static camera, we perform linear interpolation on the remaining gaps. These sequences can be automatically detected using DeepMatching on the regions outside detection boxes.

To demonstrate the performance using traditional post-processing, we also evaluate our tracker LifT using only linear interpolation as post-processing in all sequences. This method is called LifTsimInt in the tables.

4.7.3 COST LEARNING

Costs for base edges E and lifted edges E' are computed equally since they both indicate whether two detections are from the same object or not. We denote by $\text{wi}(v)$ the width of the detection bounding box corresponding to node v.

Visual cues. We exploit two different appearance features: Given two detections, the *re-identification* descriptor utilizes global appearance statistics, while the *deep-matching* descriptor relies on fine-grained pixel-wise correspondences.

We employ the state-of-the-art re-identification network (Zheng *et al.*, 2019) and train it on MOT17 train set (Milan *et al.*, 2016) together with additional re-identification datasets (Zheng *et al.*, 2015; Wei *et al.*, 2018; Ristani *et al.*, 2016b). The obtained feature value $\mathfrak{f}_{\text{re-id}}(e) \in [-1, 1]$ is modified in order to better reflect the uncertainty of a connection. We truncate values smaller 0 (corresponding to improbable connections) and re-scale the rest. First, we normalize scores between each detection v and all detections in every time frame V_j through the score of the most probable connecting edge vw. Second, all other connections than vw are downscaled.

Our second visual cue utilizes DeepMatching (DM) proposed by Weinzaepfel *et al.* (2013) to establish pixel-wise correspondences between two images. It thus serves as a reliable tracking feature (Tang *et al.*, 2016; Henschel *et al.*, 2018, 2019b).

We apply DM between boxes in two images and compute the DM intersection over union (Tang *et al.*, 2016; Henschel *et al.*, 2018) w.r.t. the whole detection boxes and on five subboxes (left/right, upper/middle/lower part). In addition, we measure for all points in a given subbox whether their matched endpoints are in the corresponding subbox again or not. This gives two additional error measures for deviation in x and y-directions. Thus, in total, we obtain a feature vector $\mathfrak{f}_{\text{DM}}(e) \in [0,1]^8$. In order to assess the reliability of DM features, the density of matching points is computed in each box and its subboxes. The smaller value is chosen for each box pair. This results in feature $\rho \in [0,1]^6$.

Motion constraints. We penalize for improbable motions by comparing the maximal displacement of DM endpoints within the sequence with the displacements of detection boxes. Assignment hypotheses of pairs of boxes representing improbable motions are penalized with a large cost.

Spatio-temporal cues. Our spatio-temporal cues utilize a simple motion compensation by computing the median DM displacement between correspondences of the background.

We assume a linear motion model, similar to Ristani and Tomasi (2018) and penalize deviations of detections from the estimated motion trajectory. This enforces spatio-temporally consistency of detections within one trajectory. Furthermore, we penalize improbable large person movements by relating velocities (in pixels per seconds) in horizontal direction to box width: $\mathfrak{f}_{\text{trans}}(vw) = \log(\mathfrak{v}_x(vw)/\min\{\text{wi}(v), \text{wi}(w)\})$.

Fusion of input features. We construct a neural network consisting of fully connected layers, batch normalization, and ReLU units taking the above described features and time differences as input and outputting scores for assignment hypotheses. The final layer uses a sigmoid activation function for producing a score in $[0,1]$. We refer to the supplemental material of Horňáková *et al.* (2020) for the exact structure of the neural network and details about the training procedure.

4.7.4 EXPERIMENT SETUP

In order to assess the suitability of the proposed lifted disjoint paths formulation for MOT, we conduct extensive experiments on three challenging benchmarks: MOT15 (Leal-Taixé *et al.*, 2015), MOT16 and MOT17 (Milan *et al.*, 2016), resulting in 39 test sequences. The sequences are filmed from static and moving cameras. While MOT16 and MOT17 share the same sequences, MOT17 provides three different detectors in order to study the dependence of the tracking quality on the input detections. We perform analysis and parameter tuning for our tracker LifT on the MOT17 train set, even when it is applied to the MOT15 sequences to ensure that it is not prone to overfitting. We follow the MOT challenge protocol and use the detections provided by the respective benchmarks. All experiments on the training set are evaluated using a leave-one-out cross-validation. This includes all of our training procedures, in particular also the training of the re-identification network.

To measure the tracking quality, the multiple object tracking accuracy (MOTA) (Bernardin and Stiefelhagen, 2008a) and the IDF1 metric (Ristani *et al.*, 2016b) are regarded as the most meaningful ones. The first incorporates the number of false negatives (FN), false positives (FP), and identity switches (IDS), thereby focusing on the coverage of persons. The latter assesses the consistency w.r.t. identities. Further tracking metrics (MT, ML) are defined in Li *et al.* (2009).

4.7.5 THE BENEFIT OF LONG-RANGE EDGES

We investigate the importance of using long-range information for MOT. To this end, we apply our proposed tracker on the MOT17 training sequence with varying maximal time gap, for which base and lifted edges are created between nodes. In order to assess the influence of the time gap on the tracking quality, we measure the *assignment* quality in terms of the MOTA and IDF1 metrics, without performing any inter- or extrapolation. To assess how well the *assignment part* is solved by our tracker, we compute the maximum achievable metrics given the filtered input detections and admissible assignment hypotheses within maximal time gaps. For this, ground truth trajectories are used. A detailed description of how we obtain the optimal assignments are given in the Appendix of Horňáková *et al.* (2020). From the result in Table 4.1, we see essentially constant MOTA scores. This is due to the fact that selecting correct connections does not change MOTA significantly except after inter- and extrapolation (which we have excluded in Table 4.1). However, we see a significant improvement in the IDF1 score, which directly penalizes wrong connections. Here, long-range edges help greatly. Moreover, both metrics, ID precision and ID recall, clearly increase with the increasing time gap. This shows that improvements by incorporating more temporal information come from using longer skip edges (impact on IDR) but most importantly, precision increases greatly. This means that ID switches are avoided thanks to lifted edges. Furthermore, the experiment shows that our designed features together with the lifted disjoint paths formulation (LDP) are well-suited for the MOT problem delivering nearly optimal assignments.

4.7.6 BENCHMARK EVALUATIONS

Finally, we compare our tracking performance on the tests sets of MOT15, MOT16 and MOT17 benchmarks with all trackers listed on the MOTChallenge which have been peer-reviewed and correspond to published work. The three benchmark datasets consist of 11/7/7 training and test sequences for MOT15/16/17 respectively. They are the standard benchmark datasets for MOT. The results in Table 4.2 show the

	0.3s	0.5s	1s	1.5s	2s	∞
MOTA (LifT)↑	52.6	52.7	52.8	52.8	**52.8**	-
MOTA (optimal)↑	53.0	53.1	53.3	53.3	**53.4**	**53.4**
IDF1 (LifT) ↑	55.7	57.8	61.8	63.8	**64.3**	-
IDF1 (optimal)↑	56.0	58.6	63.2	65.7	66.8	**69.9**
IDP (LifT) ↑	79.8	82.9	88.5	91.4	**92.1**	-
IDP (optimal)↑	80.4	84.2	90.8	94.3	95.9	**100.0**
IDR (LifT) ↑	42.7	44.5	47.4	49.0	**49.4**	-
IDR (optimal)↑	42.9	45.0	48.5	50.4	51.3	**53.4**

Table 4.1: Assignment quality of LifT without interpolation or extrapolation on the MOT17 train set with different maximal time gaps in seconds. Rows 1,3,5 and 7 show the results by LifT, rows 2,4,6, and 8 show the maximally achievable bounds with admissible assignment hypotheses up to the specified time gap. Bold numbers represent the best values per row.

tracking performance of our tracker LifT together with the best 5 performing trackers, accumulated over all sequences of the respective benchmarks. The evaluations show that we outperform all tracking systems by a large margin on all considered benchmarks. On MOT17, we improve the MOTA score from 53.5 to 60.5 and the IDF1 score from 52.3 to 65.6, which corresponds to an improvement of 13% in terms of MOTA and almost 25% in terms of the IDF1 score, indicating the effectiveness of the lifted edges. We observe similar improvements across all three benchmarks. These results reflect the near-optimal assignment performance observed on the MOT17 train set in Section 4.7.5. Finally, using only simple linear interpolation as post-processing (LifTsimInt), our tracker achieves 58.2 MOTA and 65.2 IDF1. Even then, our system clearly outperforms existing tracking systems. On average, the ILP solver needs 26.6 min. per sequence. Detailed runtimes are available in Table 4.4.

4.7.7 ABLATION STUDY ON POST-PROCESSING METHODS.

Solving the proposed lifted disjoint paths problem establishes the assignment of input detections to object identities very close to the best possible assignment (Section 4.7.5).

To localize tracked objects also in the frames in which the object detector failed to detect them, some trackers apply an additional object detector on these frames based on the available input detections. This can be seen as performing interpolation and extrapolation, if viewed from the perspective of data association in a tracking-by-detection framework, e.g. see Bergmann *et al.* (2019). As a result, improvements can be achieved from extending trajectories to image areas without input detections by applying an accurate object detector.

In order to make our tracking performance comparable with other trackers, we follow this strategy and employ an inter- and extrapolation based on Bergmann *et al.* (2019).

During the inter- and extrapolation, output detections (coming from the lifted disjoint paths solver) are preserved. In particular, the detections are not rejected, reshaped,

	Method	MOTA↑	IDF1↑	MT↑	ML↓	FP↓	FN↓	IDS↓	Frag↓
MOT17	LifT (ours)	**60.5**	**65.6**	27.0	33.6	14966	**206619**	1189	3476
	LifTsimInt (ours)	58.2	65.2	**28.6**	33.6	16850	217944	**1022**	**2062**
	Tracktor17	53.5	52.3	19.5	36.6	**12201**	248047	2072	4611
	JBNOT	52.6	50.8	19.7	35.8	31572	232659	3050	3792
	FAMNet	52.0	48.7	19.1	**33.4**	14138	253616	3072	5318
	eTC17	51.9	58.1	23.1	35.5	36164	232783	2288	3071
	eHAF17	51.8	54.7	23.4	37.9	33212	236772	1834	2739
MOT16	LifT (ours)	**61.3**	**64.7**	**27.0**	34.0	4844	**65401**	389	1034
	LifTsimInt (ours)	57.5	64.1	25.4	**34.7**	4249	72868	**335**	604
	Tracktor16	54.4	52.5	19.0	36.9	**3280**	79149	682	1480
	NOTA	49.8	55.3	17.9	37.7	7248	83614	614	1372
	HCC	49.3	50.7	17.8	39.9	5333	86795	391	**535**
	eTC	49.2	56.1	17.3	40.3	8400	83702	606	882
	KCF16	48.8	47.2	15.8	38.1	5875	86567	906	1116
2D MOT15	LifT (ours)	**52.5**	**60.0**	**33.8**	**25.8**	6837	**21610**	730	1047
	LifTsimInt (ours)	47.2	57.6	27.0	29.8	7635	24277	**554**	**803**
	Tracktor15	44.1	46.7	18.0	26.2	6477	26577	1318	1790
	KCF	38.9	44.5	16.6	31.5	7321	29501	720	1440
	AP_HWDPL_p	38.5	47.1	8.7	37.4	**4005**	33203	586	1263
	STRN	38.1	46.6	11.5	33.4	5451	31571	1033	2665
	AMIR15	37.6	46.0	15.8	26.8	7933	29397	1026	2024

Table 4.2: We compare our tracker LifT with the five best performing competing trackers w.r.t. MOTA from the MOT challenge. References: Tracktor (Bergmann *et al.*, 2019), JBNOT (Henschel *et al.*, 2019b), FAMNet (Chu and Ling, 2019), eTC (Wang *et al.*, 2019b), eHAF (Sheng *et al.*, 2018), NOTA (Chen *et al.*, 2019), HCC (Ma *et al.*, 2018), KCF (Chu *et al.*, 2019), AP_HWDPL_p (Chen *et al.*, 2017), STRN (Xu *et al.*, 2019) and AMIR15 (Sadeghian *et al.*, 2017). In addition, we compare the results to our tracker LifTsimInt that uses only a simple interpolation method (linear interpolation) as post-processing in all sequences. We outperform competing solvers on most metrics on all three MOT Challenge benchmarks, using LifT and LifTsimInt. Arrows indicate whether low or high metric values are better.

neither are their labels changed by Tracktor. Instead, we apply Tracktor to recover further locations of an object in the frames where detections of the object were missing. The procedure is based on its trajectory obtained from the lifted disjoint paths solver.

Table 4.3 reports the influence of employing inter- and extrapolation. The first two rows repeat values from Table 4.1 given the maximal 2s time gap. Since our solver produces nearly optimal data assignment with respect to the used input detections, further improvements can only be achieved by applying interpolation and extrapolation on the tracks obtained by the solver.

We compare the visual interpolation (VI) as well as visual extrapolation (VE), both using the method of Bergmann *et al.* (2019) with spatial interpolation (SI). For SI, we employ linear interpolation based solely on the geometric bounding box information.

Method	MOTA	IDF1
Assignment	52.8	64.3
Assignment (optimal)	53.4	66.8
Assignment+SI	57.8	67.6
Assignment+SI*	59.5	68.9
Assignment+VI	59.6	68.5
Assignment+VI+VE	65.7	71.5
Assignment+VI+VE+SI	**67.0**	72.4

Table 4.3: Ablation study on inter- and extrapolation, evaluated on the MOT17 train set. SI = spatial interpolation only on sequences filmed from a static camera, SI* = spatial interpolation on all sequences, VI = visual interpolation, VE = visual extrapolation. Assignment and assignment (optimal) denote the results of the lifted disjoint paths problem and the optimal assignment, as reported in Section 4.7.5 given 2s time gap. Note that Tracktor's object detector is fine-tuned on MOT17Det. In our experiments, this resulted in bigger improvements on the MOT17 training set than on the test set, compare Table 4.2.

The interpolation SI is applied only to sequences with a fixed camera in order to guarantee robust approximations. Still, the improvement by Assignment+SI over the baseline is evident. Especially the MOTA metric, which measures mainly the coverage of objects by detections, improves by about 10%. We also evaluate spatial interpolation for all sequences (SI*), which improves the tracker further to 59.5 MOTA and 68.9 IDF1. However, performing spatial interpolation on sequences with moving cameras can lead to error propagation. Thus, our final tracker LifT relies on the more robust visual interpolation and employs spatial interpolation only on sequences filmed from a static camera.

On the contrary, the visual interpolation based on Bergmann *et al.* (2019) can be applied robustly to all sequences, but only in situations where the object is visible. Accordingly, the method Assignment+VI further improves over the baseline, as it is applied to more frames.

Recovering the position of tracked objects also outside of the time range of its computed trajectory (Assignment+VI+VE) further helps to improve the tracking accuracy, enhancing MOTA by about 20% and IDF1 by about 10% IDF1, as VE extends computed trajectory thereby achieving longer identity consistencies.

Finally, we employ spatial interpolation on the remaining cases where detections are missing and the objects are fully occluded (Assignment+VI+VE+SI) resulting in a slight improvement over Assignment+VI+VE.

Note that we use the method Assignment+VI+VE+SI to evaluate LifT on the MOT15, MOT16, and MOT17 test set, as reported in Table 4.2. The impact of the post-processing on the training set using Tracktor seems to be very high. We conjectured this might be due to the fact that Tracktor's object detector is trained on MOT17Det (which are the detections of MOT17), leading to some degree of overfitting. Note that Tracktor is not trained on the MOT17 tracking ground truth so that it is still regarded as a meaningful validation procedure Bergmann *et al.* (2019). Therefore, we created another tracker LifTsimInt that uses a simple interpolation, namely only linear interpolation between detections of a trajectory, for all sequences. The tracker thus corresponds to Assignment+SI*. Comparing Table 4.3 with Table 4.2, we see that indeed, the impact of applying Tracktor during post-processing on the test set is significantly lower. We conclude that while the post-processing improves the tracking performance, the main performance of our tracker is due to our contributions.

Recall that most offline tracking systems obtain trajectories by solving a data association problem, e.g. Henschel *et al.* (2018); Tang *et al.* (2017); Ristani and Tomasi (2018). Our proposed tracker is able to achieve near-optimal results with respect to the input detections. Applying interpolation and extrapolation further improves the results, and makes it conceptually comparable to Tracktor. Still, with post-processing on our computed data association, we improve over Tracktor by 25%. We argue that solving the data association accurately is important to obtain a final high-quality result after post-processing.

	Sequence	MOTA↑	IDF1↑	MT↑	ML↓	FP↓	FN↓	IDS↓	Frag↓	S. time↓
MOT17-Train	MOT17-02-DPM	40.5	50.3	13	29	19	11017	26	23	127
	MOT17-04-DPM	69.9	73.9	41	22	298	13986	38	41	1521
	MOT17-05-DPM	58.2	67.0	31	40	40	2824	27	65	36
	MOT17-09-DPM	72.9	71.6	14	1	58	1370	15	7	59
	MOT17-10-DPM	67.4	70.2	26	8	106	4043	39	82	173
	MOT17-11-DPM	67.3	73.9	24	26	55	3017	11	28	115
	MOT17-13-DPM	63.6	67.2	45	36	64	4127	43	48	59
	MOT17-02-FRCNN	47.4	57.2	15	22	89	9656	26	27	229
	MOT17-04-FRCNN	67.5	74.1	38	21	98	15310	29	13	1535
	MOT17-05-FRCNN	60.2	68.9	35	36	73	2651	30	62	92
	MOT17-09-FRCNN	71.5	72.9	14	1	54	1451	10	7	51
	MOT17-10-FRCNN	73.2	76.2	33	2	270	3096	73	145	398
	MOT17-11-FRCNN	73.1	78.8	32	18	82	2436	18	27	133
	MOT17-13-FRCNN	77.1	75.8	68	10	203	2394	73	89	388
	MOT17-02-SDP	55.0	61.3	16	16	65	8236	52	50	586
	MOT17-04-SDP	77.7	81.8	46	13	243	10296	49	66	4133
	MOT17-05-SDP	64.0	69.5	41	22	105	2351	33	84	80
	MOT17-09-SDP	73.0	73.0	14	1	69	1356	12	12	127
	MOT17-10-SDP	75.0	78.6	35	2	349	2759	105	160	756
	MOT17-11-SDP	74.4	78.4	36	14	115	2277	27	36	198
	MOT17-13-SDP	70.8	71.4	62	24	200	3150	55	81	364
	MOT17-Train	67.0	72.4	679	364	2655	107803	791	1153	11430
MOT17-Test	MOT17-01-DPM	48.3	58.1	8	11	68	3258	10	19	38
	MOT17-03-DPM	73.3	70.1	82	17	3560	24276	160	256	24311
	MOT17-06-DPM	58.1	64.7	61	77	178	4728	28	155	113
	MOT17-07-DPM	44.4	52.3	7	21	155	9176	60	209	297
	MOT17-08-DPM	34.7	47.4	18	37	254	13507	32	44	146
	MOT17-12-DPM	48.3	62.3	18	41	35	4437	11	52	68
	MOT17-14-DPM	36.1	48.8	12	77	268	11449	91	239	323
	MOT17-01-FRCNN	47.7	58.1	8	10	246	3119	7	24	79
	MOT17-03-FRCNN	72.2	71.8	71	17	2664	26277	124	250	11678
	MOT17-06-FRCNN	60.4	63.7	68	61	279	4358	32	207	203
	MOT17-07-FRCNN	44.0	54.9	8	20	279	9110	63	227	281
	MOT17-08-FRCNN	31.9	43.3	17	37	383	13973	35	59	130
	MOT17-12-FRCNN	47.3	58.0	16	43	37	4521	11	34	84
	MOT17-14-FRCNN	36.2	49.0	16	72	629	11061	108	358	359
	MOT17-01-SDP	47.8	57.8	9	10	346	3008	10	31	95
	MOT17-03-SDP	78.2	77.3	92	13	3778	18879	132	323	16219
	MOT17-06-SDP	60.3	65.1	67	64	305	4345	33	217	144
	MOT17-07-SDP	45.8	55.0	8	18	285	8793	71	280	483
	MOT17-08-SDP	34.8	47.7	18	34	429	13288	48	69	202
	MOT17-12-SDP	47.3	60.7	18	42	158	4394	14	53	85
	MOT17-14-SDP	38.3	51.4	15	69	630	10662	109	370	376
MOT16	MOT16-01	48.3	58.2	8	10	78	3217	10	19	38
	MOT16-03	73.0	69.9	80	17	3732	24329	159	310	24311
	MOT16-06	58.2	64.7	62	77	249	4548	29	159	113
	MOT16-07	45.6	53.4	7	16	189	8637	57	212	297
	MOT16-08	43.4	55.7	18	24	284	9149	32	44	146
	MOT16-12	50.2	64.0	18	37	44	4072	11	51	68
	MOT16-14	36.1	48.8	12	77	268	11449	91	239	323
2D MOT15	ADL-Rundle-1	39.6	60.8	13	2	2277	3303	44	175	325
	ADL-Rundle-3	59.2	69.9	23	7	902	3217	29	42	153
	AVG-TownCentre	61.8	67.3	96	33	417	2217	99	213	20
	ETH-Crossing	57.6	69.3	7	9	35	387	3	18	2
	ETH-Jelmoli	51.4	67.1	18	14	520	701	12	44	20
	ETH-Linthescher	53.7	62.2	42	98	318	3795	21	95	11
	KITTI-16	36.2	32.7	5	1	456	521	108	60	57
	KITTI-19	43.3	49.4	11	17	467	2315	249	142	135
	PETS09-S2L2	56.9	43.6	9	2	476	3531	152	225	180
	TUD-Crossing	88.0	90.9	11	0	64	62	6	13	13
	Venice-1	45.8	62.1	9	3	905	1561	7	20	30

Table 4.4: We provide the results of our tracker LifT evaluated per sequence. In addition, we provide the time necessary to solve the corresponding lifted disjoint paths problem instance (S. time), in seconds. Arrows indicate whether low or high metric values are better. Tracking results on the test sets were evaluated by the MOTChallenge server https://www.motchallenge.net

5 AN EFFICIENT APPROXIMATE SOLVER FOR LIFTED DISJOINT PATHS: MAKING HIGHER ORDER MOT SCALABLE

5.1 INTRODUCTION

In Chapter 4, we generalize (DP) to lifted disjoint paths (LDP) by using additional connectivity priors in terms of lifted edges. This makes the formulation much more expressive while it maintains the feasibility set of the (DP). The optimization problem enables to take into account pairwise costs between arbitrary detections belonging to one trajectory. It thus enables to incorporate long-range temporal interactions effectively and leads to considerable improvement of recall and precision. While the integration of the global context by (LDP) is crucial to obtain high-quality tracking results, it makes the data association problem NP-hard. Still, we presented in Chapter 4 a global optimal LDP solver usable for semi-crowded sequences with reasonable computational effort. However, when applied to longer and crowded sequences, this approach is not tractable anymore, due to too high demands on runtime and memory.

In order to close this gap, we present the first approximate ILP solver for (LDP). The resulting tracker scales to big problem instances and incorporates global context with similar accuracy as LifT which uses the globally optimal LDP solver. Moreover, our approximate solver outputs certificates in terms of primal/dual gaps.

Our solver is based on a Lagrangean (dual) decomposition of the problem. This dual is iteratively optimized by dual block coordinate ascent (a.k.a. message passing) using techniques from Swoboda *et al.* (2017a), see Section 5.3.1. The decomposition relies on subproblems that are added in a cutting plane fashion. We obtain high-quality primal solutions by solving minimum cost flow problems with edge costs synthesizing information from both base and lifted edges from the dual task and improve them via a local search procedure.

The contribution of this work is in summary as follows:

- We make the (LDP) problem more accessible and applicable by introducing an approximate solver with better scalability properties than the global optimal LDP solver, while resulting in similar tracking performance, and being independent of Gurobi (Gurobi Optimization, 2019).

- We present an MOT system that is scalable to challenging sequences by using considerably less computationally demanding features than what is used in the tracker from Chapter 4. The system presented here incorporates higher order consistencies in a scalable way, i.e. it uses an approximate solver and provides a gap to the optimum. We make our LDP solver https://github.com/LPMP/LPMP and our MOT pipeline https://github.com/TimoK93/ApLift available.

5.2 RELATED WORK

We employ in our LDP solver the message passing framework from Swoboda *et al.* (2017a). It has been used for solving (MC) by Swoboda and Andres (2017), graph matching by Swoboda *et al.* (2017b), and multi-graph matching by Swoboda *et al.* (2019). It was also used within an end-to-end trainable framework for graph matching (Rolínek *et al.*, 2020).

Although Swoboda *et al.* (2017a) offered a general-purpose message passing framework for ILPs, the subproblem decomposition, reparametrization procedures, and techniques for obtaining primal solutions still have to be designed specifically for each task. Therefore, developing these procedures for (LDP) is an important contribution for solving it in a scalable way while keeping a small gap to an optimum. Lange and Swoboda (2021) proposed a parallel message passing solver based on binary decision diagrams that is usable for any 0/1 ILP problem and thus represents a step towards an efficient universal message passing approach for solving 0/1 ILPs.

Algorithms implementing dual block coordinate ascent for energy optimization are often used for solving the max-sum problem, also known as finding maximum-a-posteriori (MAP) configuration of Markov random fields. The task is to maximize a sum of unary and binary functions of discrete variables. These algorithms optimize the dual problem to the LP relaxation of the max-sum problem, i.e. minimize its upper bound. Werner (2007) reviewed the known methods and theoretical findings including the relation of this problem to constraints satisfaction programming. One of the methods called sequential tree-reweighted message passing (TRW-S) was presented by Kolmogorov (2006). Werner (2009) studied a generalized version of the problem with n-ary constraints. Kolmogorov (2014) proposed a new family of message passing techniques for MAP estimation in graphical models which can be viewed as a generalization of TRW-S from pairwise to higher-order graphical models. Kappes *et al.* (2015a) provided an empirical comparison of more than 27 state-of-the-art optimization techniques for energy optimization on discrete graphical models including those based on optimizing the dual bound for the above-stated problems. An extensive overview of the techniques for MAP inference in discrete graphical models including dual block coordinate ascent methods is provided by Savchynskyy *et al.* (2019).

We proved in our work Horňáková *et al.* (2020) (Chapter 4) that (LDP) is NP-hard in general. Ganian *et al.* (2021) inspected its parameterized complexity and provided more accurate complexity bounds if some instance parameters are fixed.

We validated the quality of our solver on four standard MOT benchmarks (Section 4.7). We achieved comparable or better performance w.r.t. the state-of-the-art trackers (at the time of the method publication) including our tracker LifT on MOT15/16/17 (Leal-Taixé *et al.*, 2015; Milan *et al.*, 2016). Furthermore, our proposed tracker ApLift performed on par with state-of-the-art on the more challenging MOT20 dataset (Dendorfer *et al.*, 2020) which is composed of long and crowded sequences. Lightweight features and a fast solver were crucial to perform tracking on such massive sequences. Our work presented in this Chapter thus extends the applicability of the successful (LDP) formulation to a wider range of instances.

5.3 LAGRANGE DECOMPOSITION ALGORITHM FOR LDP

Below we recapitulate Lagrange decomposition and the message passing primitive used in our algorithm (Section 5.3.1). Then, we propose a decomposition of the (LDP) problem into smaller but tractable subproblems (Sections 5.3.2-5.3.4). This decomposition is a dual task to an LP relaxation of (LDP). Therefore, it provides a lower bound that is iteratively increased by the message passing. We solve the (LDP) problem in a simplified version of Lagrange decomposition framework developed by Swoboda *et al.* (2017a). Our heuristic for obtaining primal solutions uses the dual costs from the subproblems (Section 5.3.6).

5.3.1 LAGRANGE DECOMPOSITION

We have an optimization problem $\min_{x \in \mathcal{X}} \langle c, x \rangle$ where $\mathcal{X} \subseteq \{0,1\}^n$ is a feasible set and $c \in \mathbb{R}^n$ is the objective vector. Its Lagrange decomposition is given by a set of *subproblems* $\mathcal{S}$ with associated *feasible sets* $\mathcal{X}^{\mathsf{s}} \subseteq \{0,1\}^{\mathsf{d}(\mathsf{s})}$ for each $\mathsf{s} \in \mathcal{S}$. We denote by $\mathsf{d}(\mathsf{s})$ the length of the vectors in $\mathcal{X}^{\mathsf{s}}$. Each coordinate i of $\mathcal{X}^{\mathsf{s}}$ corresponds to one coordinate of $\mathcal{X}$ via an injection $\pi_{\mathsf{s}} : [\mathsf{d}(\mathsf{s})] \to [n]$ alternatively represented by a matrix $A^{\mathsf{s}} \in \{0,1\}^{\mathsf{d}(\mathsf{s}),n}$ where $(A^{\mathsf{s}})_{ij} = 1 \Leftrightarrow \pi_{\mathsf{s}}(i) = j$. For each pair of subproblems $\mathsf{s}, \mathsf{s}' \in \mathcal{S}$ that contain a pair of coordinates i, j such that $\pi_{\mathsf{s}}(i) = \pi_{\mathsf{s}'}(j)$, we have a *coupling constraint* $x^{\mathsf{s}}_i = x^{\mathsf{s}'}_j$ for each $x^{\mathsf{s}} \in \mathcal{X}^{\mathsf{s}}$, $x^{\mathsf{s}'} \in \mathcal{X}^{\mathsf{s}'}$.

We require that every feasible solution $x \in \mathcal{X}$ is feasible for the subproblems, i.e. $\forall x \in \mathcal{X}, \forall \mathsf{s} \in \mathcal{S} : A^{\mathsf{s}} x \in \mathcal{X}^{s}$.

We require that the objectives of subproblems are equivalent to the original objective, i.e. $\langle c, x \rangle = \sum_{s \in \mathcal{S}} \langle \theta^s, A^{\mathsf{s}} x \rangle \; \forall x \in \mathcal{X}$. Here, $\theta^{\mathsf{s}} \in \mathbb{R}^{\mathsf{d}(\mathsf{s})}$ defines the *objective* of subproblem s.

The *lower bound* of the Lagrange decomposition given the costs θ^{s} for each $\mathsf{s} \in \mathcal{S}$ is

$$\sum_{\mathsf{s} \in \mathcal{S}} \min_{x^{\mathsf{s}} \in \mathcal{X}^{\mathsf{s}}} \langle \theta^{\mathsf{s}}, x^{\mathsf{s}} \rangle \,. \tag{5.1}$$

Given coupling constraint $x^{\mathsf{s}}_i = x^{\mathsf{s}'}_j$ and $\gamma \in \mathbb{R}$, a sequence of operations of the form $\theta^{\mathsf{s}}_i \mathrel{+}= \gamma$, $\theta^{\mathsf{s}'}_j \mathrel{-}= \gamma$ is called a *reparametrization*. Here, the formula $\theta^{\mathsf{s}}_i {+}{=} \gamma$ means that we update θ^{s}_i by adding γ to it. We define the operation $-=$ analogically.

Feasible primal solutions are invariant under reparametrizations but the lower bound (5.1) is not. The optimum of the dual lower bound equals to the optimum of a convex relaxation of the original problem, see Guignard and Kim (1987).

Min-marginal message passing. Below, we describe reparametrization updates monotonically non-decreasing in the lower bound based on *min-marginals.* Given a variable x_i^{s} of a subproblem $\mathsf{s} \in S$, the associated *min-marginal* is

$$m_i^{\mathsf{s}} = \min_{x^{\mathsf{s}} \in \mathcal{X}^{\mathsf{s}} : x_i^{\mathsf{s}} = 1} \langle \theta^{\mathsf{s}}, x^{\mathsf{s}} \rangle - \min_{x^{\mathsf{s}} \in \mathcal{X}^{\mathsf{s}} : x_i^{\mathsf{s}} = 0} \langle \theta^{\mathsf{s}}, x^{\mathsf{s}} \rangle \tag{5.2}$$

i.e. the difference between the optimal solutions with the chosen variable set to 1 resp. 0.

Proposition 5.1 (Swoboda *et al.* (2017a))**.** *Given a coupling constraints $x_i^{\mathsf{s}} = x_j^{\mathsf{s}'}$ and $\omega \in [0,1]$ the following operation is non-decreasing w.r.t. the dual lower bound* (5.1)

$$\theta_i^{\mathsf{s}} \mathrel{-}= \omega \cdot m_i^{\mathsf{s}}, \qquad \theta_j^{\mathsf{s}'} \mathrel{+}= \omega \cdot m_i^{\mathsf{s}}\,. \tag{5.3}$$

The goal of reparametrization is two-fold. (i) Improving the objective lower bound to know how far our solution is from the optimum. (ii) Using reparametrized costs as the input for our primal heuristic yields high-quality primal solutions. The key components are efficient computations of (i) optima of subproblems for obtaining lower bound (5.1), (ii) constrained optima for obtaining min-marginals (5.2), and (iii) a primal heuristic using the reparametrized costs (Section 5.3.6).

5.3.2 INFLOW AND OUTFLOW SUBPROBLEMS

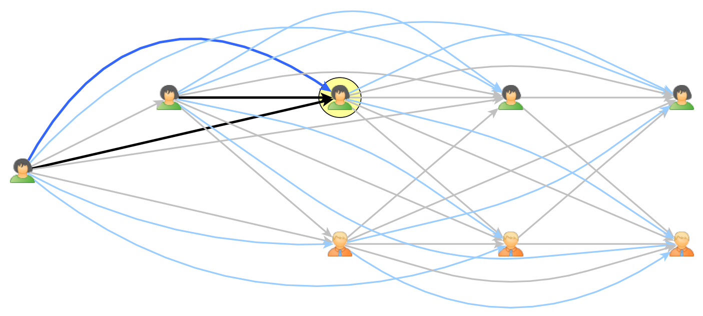
For each node $v \in V$ of the flow graph, we introduce two subproblems: An inflow and an outflow subproblem. The subproblems contain all incoming resp. outgoing edges of node v together with the corresponding node. Formally, inflow resp. outflow subproblems contain the edges $\delta_E^-(v) \cup \delta_{E'}^-(v)$, resp. $\delta_E^+(v) \cup \delta_{E'}^+(v)$. Here, we adopt the standard notation where $\delta_E^-(v)$, resp. $\delta_E^+(v)$ denote all base edges incoming to v, resp. outgoing from v. Similarly, $\delta_{E'}^-(v), \delta_{E'}^+(v)$ denote lifted edges incoming to, resp. outgoing from v.

The feasible set $\mathcal{X}^{out(v)}$ of the outflow subproblem for node v is defined as

$$\left\{ \begin{array}{l} z_v^{out(v)} \in \{0,1\}, y^{out(v)} \in \{0,1\}^{\delta_E^+(v)}, y'^{out(v)} \in \{0,1\}^{\delta_{E'}^+(v)} : \\ \quad (z_v^{out(v)}, y^{out(v)}, y'^{out(v)}) = \mathbb{0} \ \vee \ \exists P \in vt\text{-paths}(G) \text{ s.t.} \quad \begin{array}{r} z_v^{out(v)} = 1 \\ y_{vw}^{out(v)} = 1 \Leftrightarrow vw \in P_E \\ y_{vu}'^{out(v)} = 1 \Leftrightarrow u \in P_V \end{array} \end{array} \right\}. \tag{5.4}$$

Consequently, either there is no flow going through vertex v and all base and lifted edges have label zero. Alternatively, there exists a vt-path P in G labeled by one. In this case, the base edge adjacent to v corresponding to the first edge in P is one. All lifted edges connecting v with vertices of P also have value one. All other base and lifted edges are zero. Each feasible solution of the outflow subproblem can be represented by a path vt-path P. The feasible set of the inflow subproblem $\mathcal{X}^{in(v)}$ is defined analogously.

Notation. We denote by θ^{out} resp. θ^{in} the cost vector of all base edges, lifted edges, and nodes of all outflow resp. inflow subproblems. That is $\theta^{out} \in \mathbb{R}^{|V|+(|E|-|\delta_E^+(s)|)+|E'|}$. We



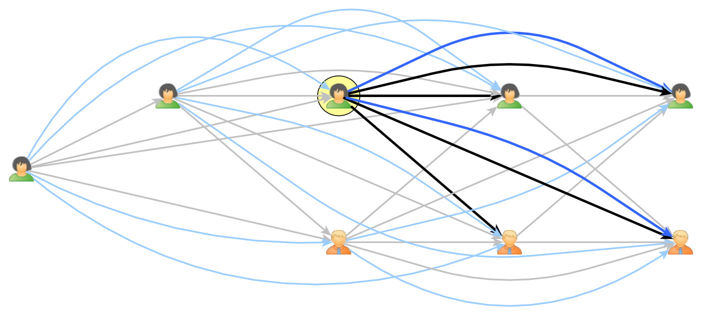


Figure 5.1: Illustration of the inflow and the outflow subproblem corresponding to the highlighted node. They contain all incoming resp. all outgoing lifted (dark blue) and base edges (black) incident to the node.

subtract the edges starting in vertex s because they are not part of any outflow subproblem. Analogically, $\theta^{in} \in \mathbb{R}^{|V|+(|E|-|\delta^-_E(t)|)+|E'|}$. We denote by $\theta^{out(v)} \in \mathbb{R}^{1+|\delta^+_E(v)|+|\delta^+_{E'}(v)|}$ the cost vector of the outflow subproblem corresponding to node v.

Note that each (lifted) edge resp. each node appears in maximally one outflow subproblem. This subproblem is uniquely identified by the first vertex of the edge resp. by the vertex itself. Therefore, we can write y^{out}_{vw} for the edge variable and θ^{out}_{vw} for the edge cost value instead of $y^{out(v)}_{vw}$ and $\theta^{out(v)}_{vw}$. This leads to better readability. Analogically, we usually write z^{out}_v and θ^{out}_v instead of $z^{out(v)}_v$ and $\theta^{out(v)}_v$ for the node variables and costs in the outflow subproblems. The cost vector of the outflow subproblem of vertex v has the following elements:

$$\theta^{out(v)} = (\theta^{out}_v, \theta^{out}_{vv_1}, \dots, \theta^{out}_{vv_n}, \theta'^{out}_{vw_1}, \dots, \theta'^{out}_{vw_m}) \tag{5.5}$$

Here $n = |\delta^+_E(v)|$ is the number of base edges and $m = |\delta^+_{E'}(v)|$ is the number of lifted edges in the subproblem. The dashed variables $\theta'^{out}_{vw_i}$ denote the costs of lifted edges. We use an analogical notation for the inflow subproblems.

Constraints between inflow and outflow subproblems. We require that all variables obey the coupling constraints in any primal solution. That is, the corresponding variables from different subproblems must have equal values. For node variables, we add the constraint $z^{in}_v = z^{out}_v$. For an edge $vw \in E \cup E'$ we require the shared edge in the outflow subproblem of v and in the inflow subproblem for w to agree, i.e. $y^{out}_{vw} = y^{in}_{vw}$ if $vw \in E$ and $y'^{out}_{vw} = y'^{in}_{vw}$ if $vw \in E'$. In the dual solution, however, the coupling

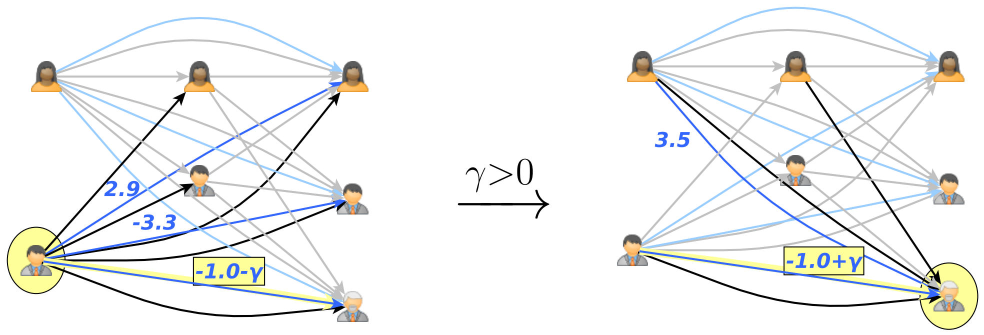


Figure 5.2: An illustration of sending message delta from an outflow subproblem to an inflow subproblem. The optimal solution of the outflow subproblem activates the lifted edge with cost −3.3, activating the highlighted lifted edge is suboptimal. Therefore, the message is positive and activating the highlighted edge in the inflow subproblem becomes less beneficial.

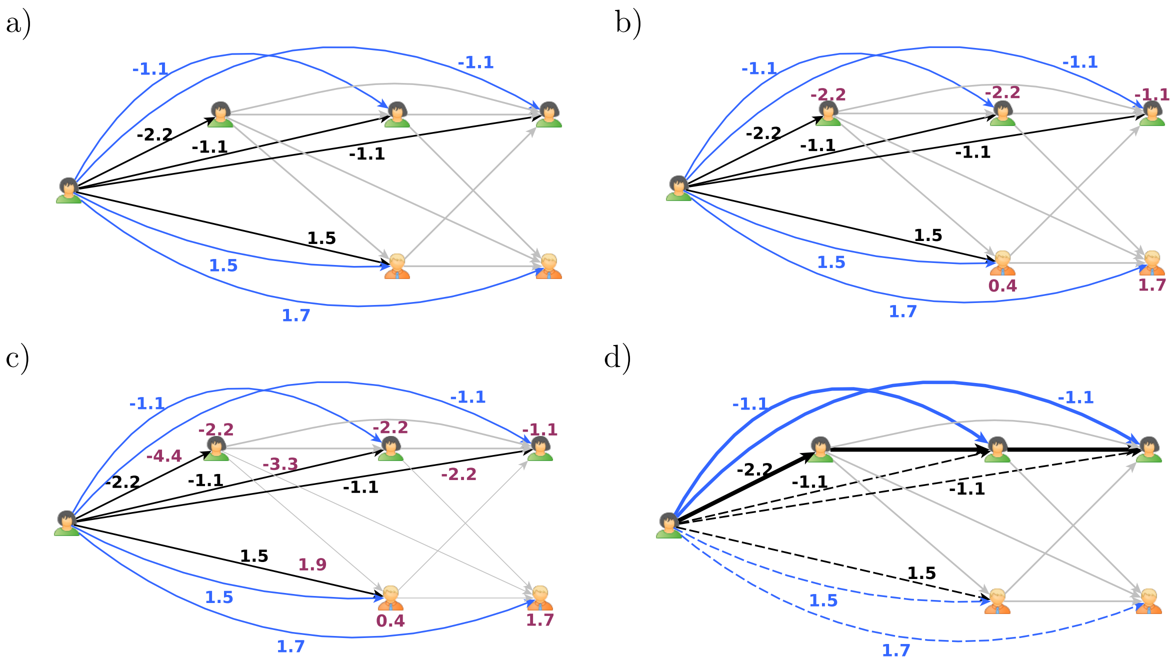


Figure 5.3: Optimization of an outflow subproblem (Algorithm Opt-Out-Cost). This method is also employed in the computation of the outflow subproblem min marginals (Algorithms All-Base-MM-Out and All-Lifted-MM-Out). Given an outflow subproblem with lifted and base edge costs (a), we first compute values in array lifted_costs for each incident node (violet numbers in (b)). These numbers must be obtained from right to left, eventually in the DFS manner (see Algorithm Lifted-Cost-DFS-Out). Given these values, we can compute for each base edge, the constrained optima where the edge is active. These are the violet labels of the base edges in (c). They represent vector α from Algorithm Opt-Out-Cost. (d) shows the optimal solution.

Algorithm Opt-Out-Cost $(v, \tilde{\theta})$

Input start vertex v, vector of costs $\tilde{\theta}$
Output optimal value opt, lifted_cost, $\forall w : vw \in \delta_E^+(v)$ optimal solution for vw active α_{vw}

```
1: for $u \in V : vu \in \mathcal{R}_G$ do
2:     $\text{lifted\_cost}[u] = \infty$, $\text{next}[u] = \emptyset$
3: end for
4: $\text{lifted\_cost}[t] = 0$, $\text{next}[t] = t$
5: Lifted-Cost-DFS-Out$(v, v, \tilde{\theta}, \text{lifted\_cost}, \text{next})$
6: $\forall w : vw \in \delta_E^+(v) : \alpha_{vw} = \tilde{\theta}_v + \tilde{\theta}_{vw} + \text{lifted\_cost}[w]$
7: $\text{opt} = \min(\min_{vw \in \delta_E^+(v)} \alpha_{vw}, 0)$
```

Algorithm Lifted-Cost-DFS-Out $(v, u, \tilde{\theta}, \text{lifted\_cost}, \text{next})$

Input $v, u, \tilde{\theta}, \text{lifted\_cost}, \text{next}$
Output $\text{lifted\_cost}, \text{next}$

```
1: $\alpha = 0$
2: for $uw \in \delta_E^+(u)$ do
3:     if $\text{next}[w] = \emptyset$ then Lifted-Cost-DFS-Out$(v, w, \tilde{\theta})$
4:     if $\text{lifted\_cost}[w] < \alpha$ then
5:         $\alpha = \text{lifted\_cost}[w]$, $\text{next}[u] = w$
6:     end if
7: end for
8: if $\text{next}[u] = \emptyset$ then $\text{next}[u] = t$
9: $\text{lifted\_cost}[u] = \alpha + \tilde{\theta}'_{vu}$
```

constraints are not required. Instead, messages are sent between the coupled variables to improve the consistency between their assignments and increase the lower bound. Figure 5.2 illustrates sending a message from the outflow subproblem to an inflow subproblem that shares an edge with it.

Optimization of in- and outflow subproblems. Given costs $\theta^{out(v)}$, the optimal solution of an outflow problem for node v can be computed by depth-first search on the subgraph defined by the vertices reachable from v. The algorithms rely on the following data structures:

- $\text{lifted\_costs}[u]$ contains the cost of the minimal ut-path w.r.t. to costs of all lifted edges connecting v with the vertices of the path.
- $\text{next}[u]$ contains the best neighbor of vertex u w.r.t. values in lifted_cost. That is, $\text{next}[u] = \text{argmin}_{w:uw \in \delta_E^+(u)} \text{lifted\_cost}[w]$.

Algorithms Opt-Out-Cost and Lifted-Cost-DFS-Out give a general depth first search (DFS) procedure that, given a vertex v, computes optimal paths from all vertices reachable from v. Algorithm Opt-Out-Cost takes as input vertex v and cost vector $\tilde{\theta} = (\tilde{\theta}_v, \tilde{\theta}_{vv_1}, \ldots, \tilde{\theta}_{vv_n}, \tilde{\theta}'_{vw_1}, \ldots, \tilde{\theta}'_{vw_m})$, as in (5.5). Its subroutine Algorithm Lifted-Cost-

DFS-Out computes recursively for each vertex u reachable from v the value lifted_cost$[u]$.

The overall optimal cost $\min_{(z,y,y')\in\mathcal{X}_v^{out}} \langle\tilde{\theta},(z,y,y')\rangle$ of the subproblem is given by the minimum of node and base edge and lifted edges costs $\min_{vu\in\delta_E^+(v)} \tilde{\theta}_v + \tilde{\theta}_{vu} + \text{lifted\_cost}[u]$. Figure 5.3 illustrates the method. We achieve linear complexity by exploiting that subpaths of minimum cost paths are minimal as well. The optimization for the inflow subproblem is analogous.

Message passing for in- and outflow subproblems. We could compute one min-marginal (5.2) by adapting Algorithm Opt-Out-Cost and forcing an edge to be taken or not. However, computing min-marginals one-by-one by performing operation (5.3) would be inefficient, since it would involve calling Algorithm Opt-Out-Cost $\mathcal{O}(|\delta_E^+(v)|+|\delta_{E'}^+(v)|)$ times. Therefore, we present efficient algorithms for computing a sequence of min-marginals in Section 5.4. The procedures save computations by choosing the order of edges for computing min-marginals suitably and reusing previous calculations. We provide remarks on how to efficiently implement all used DFS procedures in Section 5.4.1.

5.3.3 PATH SUBPROBLEMS

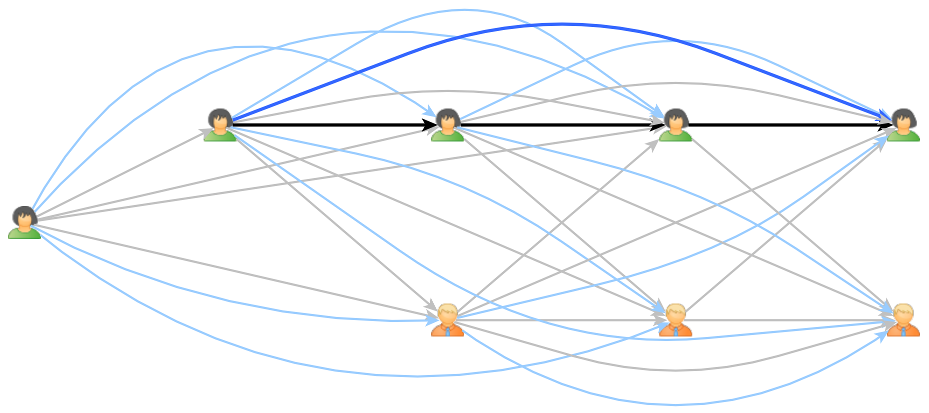


Figure 5.4: A path subproblem contains a lifted edge and a path between its endpoints that can contain lifted edges too.

A path subproblem contains a lifted edge vw and a path P from v to w consisting of both base and lifted edges (see Figure 5.4). They reflect that (i) lifted edge vw must be labeled 1 if there exists an active path between v and w, and (ii) there cannot be exactly one inactive lifted edge within path P if vw is active. The reason is that the inactive lifted edge divides P into two segments that must be disconnected. This is contradictory to activating lifted edge vw because it indicates a connection between v and w. Path subproblems are similar to cycle inequalities for the multicut (2.1).

In order to distinguish between the base and lifted edges of path P, we use notation $P_E = P\cap E$ and $P_{E'} = P\cap E'$. For the purpose of defining the feasible solutions of path subproblems, we define strong base edges $E_0 = \{vw\in E | vw\text{-paths}(G) = \{vw\}\}$. That

Algorithm Path-Subproblem-Optimization (θ^P)

```
Input Edge costs $\theta^P$
Output optimal value opt of subproblem.
 1: $E^+ = \{kl \in P_{E'} \cup vw | \theta'^P_{kl} > 0\} \cup \{kl \in P_E | \theta^P_{kl} > 0\}$
 2: if $E^+ = \{kl\} \wedge kl \in P_{E'} \cup vw \cup E_0$ then
 3:     $\alpha = \min\{\min_{ij \in P_E \setminus E^+} |\theta^P_{ij}|, \min_{ij \in P_{E'} \cup vw \setminus E^+} |\theta'^P_{ij}|\}$
 4:     $\beta = \begin{cases} \theta'^P_{kl} & \text{if } kl \in P_{E'} \cup vw \\ \theta^P_{kl} & \text{if } kl \in P_E \end{cases}$
 5:     if $\alpha < \beta$ then
 6:         opt $= \sum_{ij \in P_E \setminus E^+} \theta^P_{ij} + \sum_{ij \in P_{E'} \cup vw \setminus E^+} \theta'^P_{ij} + \alpha$
 7:     else
 8:         opt $= \sum_{ij \in P_E} \theta^P_{ij} + \sum_{ij \in P_{E'} \cup vw} \theta'^P_{ij}$
 9:     end if
10: else
11:     opt $= \sum_{ij \in P_E \setminus E^+} \theta^P_{ij} + \sum_{ij \in P_{E'} \cup vw \setminus E^+} \theta'^P_{ij}$
12: end if
13: return opt
```

is, base edge vw is strong iff there exists no other vw-path in graph G than vw itself.

The feasible set $\mathcal{X}^P$ of the path subproblem for vw-path P is defined as

$$
\begin{aligned}
& y \in \{0,1\}^{P_E}, y' \in \{0,1\}^{P_{E'} \cup \{vw\}} : \\
& \qquad \forall kl \in P_{E'} \cup \{vw\} : \sum_{ij \in P_E} (1 - y_{ij}) + \sum_{ij \in P_{E'} \cup \{vw\} \setminus \{kl\}} (1 - y'_{ij}) \geq 1 - y'_{kl} \,, \qquad (5.6) \\
& \qquad \forall kl \in P_E \cap E_0 : \sum_{ij \in P_E \setminus kl} (1 - y_{ij}) + \sum_{ij \in P_{E'} \cup \{vw\}} (1 - y_{ij}) \geq 1 - y_{kl} \,. \qquad (5.7)
\end{aligned}
$$

Equation (5.6) requires that a lifted edge in $P_{E'}$ or vw can be zero only if at least one other edge of the subproblem is zero. Equation (5.7) enforces the same for strong base edges.

The optimization of path subproblems. We denote by θ^P the edge costs in the subproblem of vw-path P. The optimization over the feasible set $\mathcal{X}^P$ w.r.t. costs θ^P is detailed in Algorithm Path-Subproblem-Optimization. The principle is as follows. It checks whether there exists exactly one positive edge and whether it is either a lifted or a strong base edge (Line 2). If so, the optimal solution is either (i) all edges except the two largest ones (Line 6) or (ii) all edges (Line 8), whichever gives a smaller objective value. If the above condition does not hold, the optimal solution can be chosen to contain all negative edges (Line 11).

We use a variation of the path optimization algorithm with an edge fixed to 0 or 1 for computing min-marginals.

Cutting plane. Since there are exponentially many path subproblems, we add during the optimization only those that improve the relaxation. Details are in Section 5.4.2.

5.3.4 CUT SUBPROBLEMS

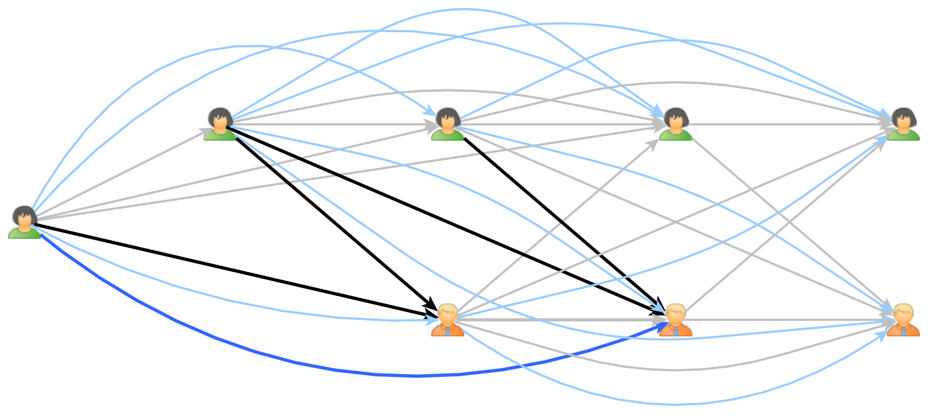


Figure 5.5: A cut subproblem consists of a lifted edge (dark blue) and a set of base edges (black) that compose a cut between the nodes of the lifted edge.

The purpose of a cut subproblem is to reflect that a lifted edge uv must be labeled 0 if there exists a cut of base edges that separate u and v (uv-cut) all labeled 0.

The feasible set. A cut subproblem consists of a lifted edge uv and a uv-cut $C = \{ij \in E | i \in A, j \in B\}$ where $A, B \subset V$ with $A \cap B = \emptyset$. The space of feasible solutions $\mathcal{X}^C$ is defined as

$$y'_{uv} \in \{0,1\}, y \in \{0,1\}^C : \quad y'_{uv} \leq \sum_{ij \in C} y_{ij}\,, \quad uv \in C \Rightarrow y'_{uv} \geq y_{uv}$$
$$\forall i \in A : \sum_{ij \in C} y_{ij} \leq 1\,, \quad \forall j \in B : \sum_{ij \in C} y_{ij} \leq 1\,. \tag{5.8}$$

The constraints stipulate that (i) the lifted edge uv is 0 if all the edges in the cut are 0, (ii) if there is also base edge $uv \in C$ then whenever it is active, the lifted edge uv must be active, and (iii) there exists at most one active outgoing resp. incoming edge for every vertex in A resp. B.

Optimization of a cut subproblem with respect to feasible set $\mathcal{X}^C$ is given by Algorithm Cut-Subproblem-Optimization. Its key is to solve a linear assignment problem (LAP) (Ahuja *et al.*, 1988) between vertex sets A and B. The assignment cost ψ_{ij} for

Algorithm Cut-Subproblem-Optimization (θ^C)

```
Input Edge costs $\theta^C$
Output optimal value opt of subproblem.
 1: Define $\psi \in \mathbb{R}^{A \times B}$:
 2: $\psi_{ij} = \begin{cases} \theta^C_{uv} + \theta'^C_{uv}, & \text{if } ij = uv \wedge uv \in C \wedge \theta'^C_{uv} > 0 \\ \infty, & \text{if } ij \notin C \\ \theta^C_{ij}, & \text{otherwise} \end{cases}$
 3: $z^* \in \operatorname{argmin}_{z \in \{0,1\}^{A \times B}} \sum_{i \in A} \sum_{j \in B} \psi_{ij} z_{ij}$, s.t. $z\mathbb{1} \le \mathbb{1}, z^\top \mathbb{1} \le \mathbb{1}$
 4: opt $= \sum_{ij \in C} \psi_{ij} z^*_{ij}$
 5: if $\theta'^C_{uv} \ge 0$ then return opt
 6: if $\exists kl \in C : z_{kl} = 1$ then
 7:     return opt $+ \theta'^C_{uv}$
 8: else
 9:     $\alpha = \min_{ij \in C} \theta^C_{ij}$
10:     if $|\theta'^C_{uv}| > \alpha$ then return $\theta'^C_{uv} + \alpha$
11:     else return opt
12: end if
```

$(i, j) \in A \times B$ is the cut edge cost θ^C_{ij} if edge ij belongs to C and ∞ otherwise. In the special case of uv-cut C containing base edge uv and the lifted edge cost θ'^C_{uv} being positive, the assignment cost ψ_{uv} is increased by θ'^C_{uv}.

A candidate optimal labeling of cut edges is given by values of LAP variables z_{ij}. If $\theta'^C_{uv} \ge 0$, the optimal value found by the LAP is the optimal value of the cut subproblem. If it is negative, we distinguish two cases: (i) If a cut edge kl labeled by one exists, the lifted edge cost θ'^C_{uv} is added to the optimal value of LAP. (ii) Otherwise, we inspect whether it is better to activate the smallest-cost cut edge and the lifted edge uv or keep all edges inactive.

We use a variation of Algorithm Cut-Subproblem-Optimization with an edge variable restricted to be either 0 or 1 for computing min-marginals.

Cutting plane. There are exponentially many cut subproblems. Therefore, we add only those that improve the lower bound. See Section 5.4.3 for details.

5.3.5 MESSAGE PASSING

The overall algorithm for optimizing the Lagrange decomposition is Algorithm Message-Passing in Section 5.4. First, inflow and outflow subproblems are initialized for every node. Then, for a number of iterations or until convergence, costs for each subproblem are adjusted iteratively by computing min-marginals and adjusting the reparametrization proportionally to the min-marginal's value. Additionally, every k-th iteration new path and cut subproblems are separated and added to the Lagrange decomposition.

Algorithm Compute-Primal $(\mathcal{S}, \theta)$

Input Subproblems $\mathcal{S}$, original costs $\theta \in \mathbb{R}^{|V'|+|E|+|E'|}$
Output Primal solution (z, y, y')

```
1: Init-MCF
2: Obtain primal solution of MCF $y^{mcf} \in \{0,1\}^{E^{mcf}}$
3: Set $(z, y)$ according to $y^{mcf}$
4: $y'$ =Adjust-Lifted-Solution$(z, y)$
5: $(z, y, y')$ =Local-Search$(z, y, y')$
```

Algorithm Init-MCF $(\theta^{in}, \theta^{out})$

Input The current variable costs in all inflow and outflow subproblems $(\theta^{in}, \theta^{out})$
Output Costs of base edges for the MCF problem θ^{mcf}

```
1: $\forall u \in V \backslash \{s, t\}$:
     $(o, lc, \alpha^{in}_u)$=Opt-In-Cost$(u, \theta^{in(u)})$
     $(o, lc, \alpha^{out}_u)$=Opt-Out-Cost$(u, \theta^{out(u)})$
2: $\forall u \in V \backslash \{s, t\} : \theta^{mcf}_{su^{in}} = \alpha^{in}_{su}, \theta^{mcf}_{u^{out}t} = \alpha^{out}_{ut}$
3: $\forall u \in \{uv \in E | u \neq s, v \neq t\} : \theta^{mcf}_{u^{out}v^{in}} = \alpha^{out}_{uv} + \alpha^{in}_{uv}$
```

5.3.6 PRIMAL ROUNDING

For computing primal solutions we solve the minimum cost flow (MCF) problem on the base edges and improve this initial solution with a local search heuristic (Algorithm Compute-Primal).

Without lifted edges, the disjoint paths problem is an instance of (MCF) , which can be efficiently optimized via combinatorial solvers like the successive shortest path solver that we employ (Ahuja *et al.*, 1988). We enforce node disjoint paths via splitting each node $u \in V$ into two nodes $u^{in}, u^{out} \in V^{mcf}$ in the MCF graph $G^{mcf} = (V^{mcf}, E^{mcf})$, adding an additional edge $u^{in}u^{out}$ to E^{mcf} and setting capacity $[0, 1]$ on all edges E^{mcf}. Each node except s and t has demand 0. Algorithm Init-MCF calculates edge costs for (MCF) from in/outflow subproblems using Algorithm Opt-Out-Cost. We obtain the cost of each flow edge $u^{out}v^{in}$ from the inflow subproblem of v and the outflow subproblem of u by summing up their minima where edge uv is active. This combines well the cost from the base and lifted edges because the minimum comprises the best feasible combination of the lifted edges that are reachable when uv is active.

Algorithm Local-Search in Section 5.4 is the local search heuristic for improving the (MCF) solution. It works with sets of disjoint paths. First, paths are split if this leads to a decrease in the objective. Second, merges are explored. If a merge of two paths is not possible, we iteratively check whether cutting off one node from the first path's end or the second path's beginning makes the connection possible. If yes and the connection is decreasing the objective, the nodes are cut off and the paths are connected.

5.4 MESSAGE PASSING ALGORITHMS

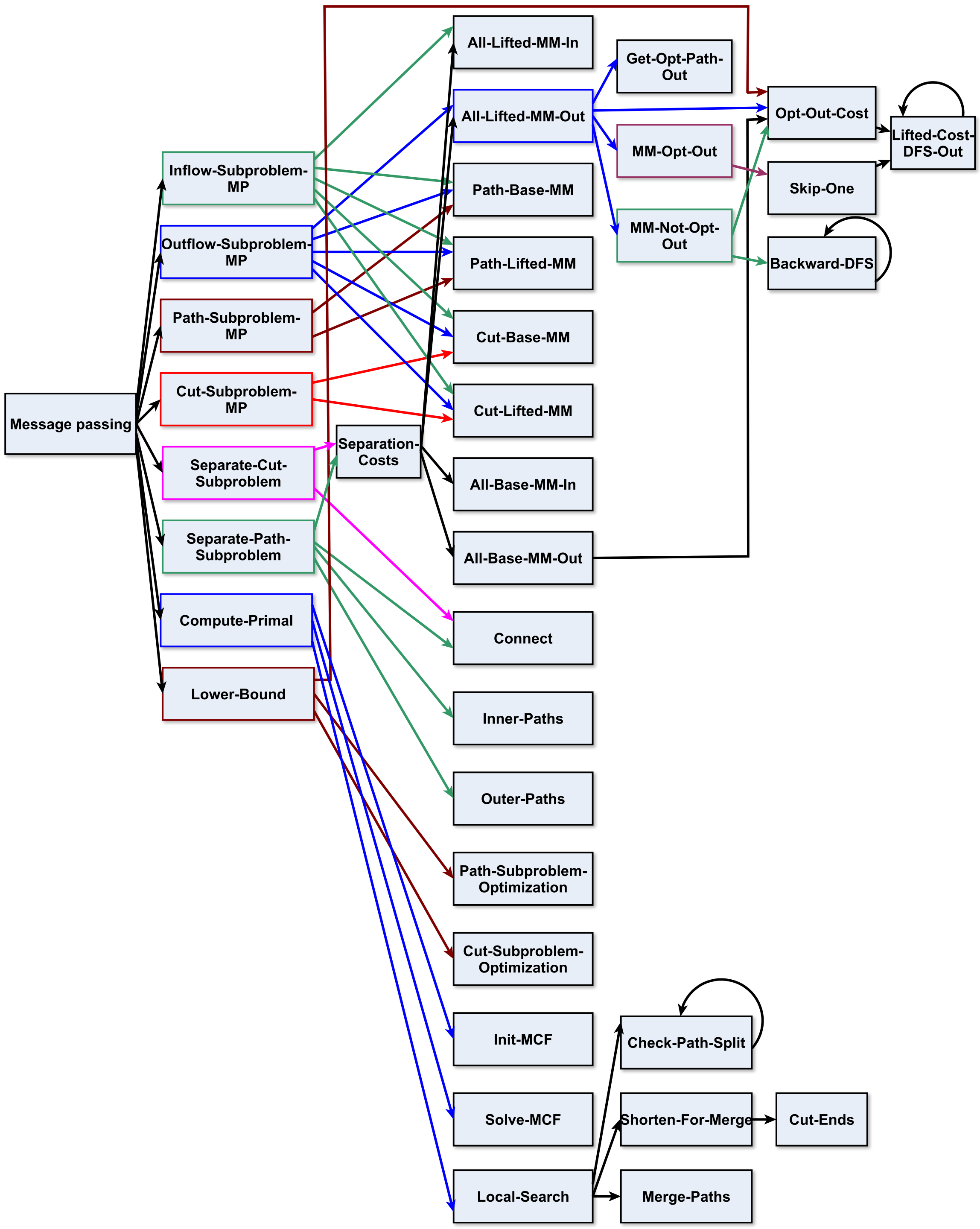


Figure 5.6: The scheme of our algorithms. Arrows from algorithms point to their subroutines. MP means Message Passing. Some inflow algorithms analogical to outflow are omitted.

Algorithm All-Base-MM-Out $(v, \tilde{\theta})$

```
Input start vertex $v$, costs $\tilde{\theta}$
Output base edge min-marginals $\gamma_{vu}$ $\forall vu \in \delta^+_E(v)$
1: $(\text{opt}, \text{lifted\_cost}, \alpha)$ =Opt-Out-cost$(v, \tilde{\theta})$
2: $e^* = \underset{vw \in \gamma^+_E}{\text{argmin}}\{\alpha_{vw}\}$, $e^{**} = \underset{vw \in \gamma^+_E \setminus \{e^*\}}{\text{argmin}}\{\alpha_{vw}\}$
3: $\forall vu \in \delta^+(v) : \gamma_{vu} = \alpha_{vu} - \min(\alpha_{e^{**}}, 0)$
```

Algorithm All-Lifted-MM-Out $(v, \tilde{\theta})$

```
Input starting vertex $v$, cost vector $\tilde{\theta}$
Output lifted edge min-marginals $\gamma'_{vu}$ $\forall vu \in \delta^+_{E'}(v)$
1: $(\text{opt}, \text{lifted\_cost}, \alpha, \text{next})$ =Opt-Out-cost$(v, \tilde{\theta})$
2: $P^*_V$=Get-Opt-Path-Out$(\tilde{\theta}, \alpha, \text{next})$
3: $\forall vw \in \delta^+_{E'}(v) : \gamma'_{vw} = 0$
4: $(\text{opt}, \gamma')$ =MM-Opt-Out$(v, P^*_V, \text{opt}, \gamma', \tilde{\theta})$
5: $\gamma'$ =MM-Not-Opt-Out$(v, \text{opt}, \gamma', \tilde{\theta})$
```

LDP solver outline. Figure 5.6 contains the scheme of all algorithms used in our approximate LDP solver. The algorithms are stated either in Section 5.3 or in this section. The solver performs an explicitly given number of message passing iterations. Section 5.4.5 describes the full solver run and an overview of all methods used within one message passing iteration. Once in five iterations, a new primal solution is computed (Sections 5.3.6 and 5.4.6). Once in twenty iterations, new subproblems are separated and added to the problem in order to tighten the LP relaxation. These are path and cut subproblems (see Sections 5.3.3 and 5.3.4). Methods for their separations are described in Sections 5.4.2 and 5.4.3. Section 5.4.4 discusses the guaranteed lower bound improvement achieved by separating the new subproblems

Message passing. Messages are sent between the subproblems. Each subproblem creates messages to be sent by computing min-marginals of its variables. Section 5.4.1 presents algorithms used for obtaining min-marginals of inflow and outflow subproblems. The algorithms allow us to efficiently obtain min-marginals of all lifted or all base edges of a subproblem at once. Messages from cut and path subproblems are obtained by modifications of the respective algorithms for their optimization.

5.4.1 MIN-MARGINALS FOR INFLOW AND OUTFLOW SUBPROBLEMS

We detail routines for computing min-marginals for all base edges at once (Algorithm All-Base-MM-Out) and all lifted edges at once (Algorithm All-Lifted-MM-Out). All the stated algorithms assume outflow subproblems. Modification to inflow subproblems is done via proceeding in the opposite edge direction.

We use these algorithms because iteratively computing min-marginals and performing

Algorithm Get-Opt-Path-Out (next, α)

Input next: best descendants w.r.t. lifted_costs, vector α such that $\forall vw \in \delta^+_E(v) : \alpha_{vw}$ is the optimal value if vw is active
Output min cost path P^*_V

```
1: $w^* = \text{argmin}_{w:vw\in\delta^+_E(v)} \alpha_{vw}$
2: if $\alpha_{vw^*} < 0$ then
3:     while $w^* \neq t$ do
4:         $P^*_V \leftarrow w^*$
5:         $w^* = \text{next}[w^*]$
6:     end while
7: else
8:     $P^*_V = \emptyset$
9: end if
```

operation (5.3) would be inefficient, since it would involve calling Algorithm Opt-Out-Cost $\mathcal{O}(|\delta^+_E(v)| + |\delta^+_{E'}(v)|)$ times. Algorithms All-Base-MM-Out and All-Lifted-MM-Out speed up iterative min-marginal updates by reusing computations.

Algorithm All-Base-MM-Out for computing base edge min-marginals uses the fact that lifted edge costs do not change and therefore Algorithm Opt-Out-Cost needs to be called only once. For lifted edges, Algorithm All-Lifted-MM-Out interleaves min-marginal computation and reparametrization updates (5.3) such that computations can be reused.

In Algorithm All-Lifted-MM-Out, path P^* representing the optimal solution of the outflow problem is found by calling Algorithm Opt-Out-Cost followed by Algorithm Get-Opt-Path-Out. We introduce auxiliary variables γ'_{vw} in Line 3 that keep track of future reparametrization updates. Then, Algorithm MM-Opt-Out computes min-marginals for the lifted edges that are active in the optimal solution. At the end of Algorithm All-Lifted-MM-Out, min-marginals are computed for those lifted edges that are not active in the optimal solution by calling MM-Not-Opt-Out.

For computing min-marginals of edges that are active in the optimal solution (Algorithm MM-Opt-Out), we need as a subroutine Algorithm Skip-One, an extended version of Algorithm Opt-Out-Cost. Algorithm Skip-One restricts the vertices taken into consideration during the optimization. In particular, a special vertex r is given that is to be excluded from the optimization. Values lifted_cost$[u]$ are reused for those vertices u where $ur \notin \mathcal{R}_G$ because these values are not affected by excluding vertex r.

Min-marginals for vertices inactive in the optimal solution are computed by Algorithms MM-Not-Opt-Out and Backward-DFS . The algorithms rely on structure back_cost which is an analogy of lifted_cost. Structure back_costs$[u]$ contains the minimum cost of all vu-paths w.r.t. to the costs of all lifted edges connecting v with the vertices of the path plus the cost of the first base edge of the path. Note that lifted_costs$[u]$ is defined analogically but contains the minimum cost of all ut-paths. Therefore, the minimal solution where a lifted edge $vu \in E'$ is active can be obtained as follows:

$$\min_{(z,y,y')\in\mathcal{X}^{out(v)}:y'_{vu}=1} \langle\tilde{\theta}, (z_v, y, y')\rangle = \text{lifted\_cost}[u] + \text{back\_cost}[u] - \tilde{\theta}'_{vu} \quad (5.9)$$

Algorithm MM-Opt-Out $(v, P^*, \text{opt}, \gamma', \tilde{\theta})$

Input starting vertex v, optimal path $P_V^* = (v_1, \dots, v_k)$, value of optimal path opt, reparametrization updates γ', costs $\tilde{\theta}$
Output updated cost of optimal path opt, new γ'

```
1: for all $v_i = v_1, \dots, v_k : vv_i \in \delta^+_{E'}(v)$ do
2:     $\alpha =$Skip-One$(v, v_i, \tilde{\theta} - (\mathbb{0}, \gamma'), \text{lifted\_cost}, \text{next})$
3:     $\gamma'_{vv_i} = \text{opt} - \alpha$
4:     $\text{opt} = \alpha$
5: end for
```

Algorithm Skip-One $(v, r, \tilde{\theta}, \text{lifted\_cost}, \text{next})$

Input v, ignored vertex r, $\tilde{\theta}$, lifted_cost, next
Output optimal value opt

```
1: for $u \in V : vu \in \mathcal{R}_G \wedge ur \in \mathcal{R}_G$ do
2:     $\text{lifted\_cost}[u] = \infty$, $\text{next}[u] = \emptyset$
3: end for
4: $\text{lifted\_cost}[r] = 0$, $\text{next}[r] = t$
5: Lifted-Cost-DFS-Out$(v, v, \tilde{\theta}, \text{lifted\_cost}, \text{next})$
6: $\forall w : vw \in \delta^+_E(v) : \alpha_{vw} = \tilde{\theta}_v + \tilde{\theta}_{vw} + \text{lifted\_cost}[w]$
7: $\text{opt} = \min(\min_{w:vw\in\delta^+_E(v)} \alpha_{vw}, 0)$
```

The cost of lifted edge $\tilde{\theta}'_{vu}$ must be subtracted because it is involved in both values lifted_cost$[u]$ and back_cost$[u]$.

Algorithm Backward-DFS performs two tasks simultaneously. First, it is a DFS procedure for computing back_cost. Contrary to Algorithm Lifted-Cost-DFS-Out that performs DFS for obtaining lifted_cost, Algorithm Backward-DFS proceeds in the opposite edge direction. It again uses the fact that a subpath of a minimum-cost path must be minimal. Second, it directly computes min marginal for already processed vertex u on Line 10 and involves this change in setting back_cost$[u]$ on Line 11.

Speeding up DFS: All the algorithms for obtaining optimal solution or min-marginals of inflow and outflow subproblems call DFS procedures. It can be considered that the order of processing the relevant nodes reachable from the central node is always the same. Therefore, we call DFS for each inflow and outflow subproblem only once during their initialization and store the obtained list of processed nodes. The full DFS in Algorithm Lifted-Cost-DFS-Out is replaced by traversing the precomputed node list in the forward direction. Algorithm Backward-DFS is replaced by traversing this node list in the backward direction.

Algorithm MM-Not-Opt-Out $(v, \text{opt}, \gamma', \tilde{\theta})$

Input v, current optimum opt, reparametrization update γ', $\tilde{\theta}$
Output changed reparametrization update γ'

```
 1: $(\text{opt}, \text{lifted\_cost}) = \mathsf{Opt\text{-}Out\text{-}cost}(v, \tilde{\theta} - (\mathbb{0}, \gamma'))$
 2: for all $u : vu \in \mathcal{R}_G$ do
 3:     if $u \in P_V^*$ then
 4:         $\text{visited}[u] = true$
 5:         $\text{back\_cost}[u] = \text{opt} - \text{lifted\_cost}[u]$
 6:         if $vu \in E'$ then $\text{back\_cost}[u] \mathrel{+}= \tilde{\theta}'_{vu} - \gamma'_{vu}$
 7:     else
 8:         $\text{visited}[u] = false$
 9:         if $vu \in \delta^+_E(v)$ then
10:             $\text{back\_cost}[u] = \tilde{\theta}_{vu}$
11:         else
12:             $\text{back\_cost}[u] = \infty$
13:         end if
14:     end if
15: end for
16: for all $vu \in \delta^+_{E'}(v)$ do
17:     if $\text{visited}[u] = false$ then
18:         $\mathsf{Backward\text{-}DFS}(v, u, \tilde{\theta}, \gamma', \text{opt}, \text{back\_cost})$
19:     end if
20: end for
```

Algorithm Backward-DFS $(v, u, \tilde{\theta}, \gamma', \text{opt}, \text{back\_cost})$

Input $v, u, \tilde{\theta}, \gamma', \text{opt}, \text{back\_cost}$
Output γ', back_cost

```
 1: $\alpha = \text{back\_cost}[u]$
 2: for $wu \in \delta^-_E(u) : vw \in \mathcal{R}_G$ do
 3:     if $\text{visited}[w] = false$ then
 4:         $\mathsf{Backward\text{-}DFS}(v, w, \tilde{\theta}, \gamma', \text{opt}, \text{back\_cost})$
 5:     end if
 6:     $\alpha = \min\{\text{back\_cost}[w], \alpha\}$
 7: end for
 8: if $vu \in E'$ then
 9:     $\text{opt}_u = \alpha + \text{lifted\_cost}[u]$
10:     $\gamma'_{vu} = \text{opt}_u - \text{opt}$
11:     $\text{back\_cost}[u] = \alpha + \tilde{\theta}'_{vu} - \gamma'_{vu}$
12: else
13:     $\text{back\_cost}[u] = \alpha$
14: end if
15: $\text{visited}[u] = true$
```

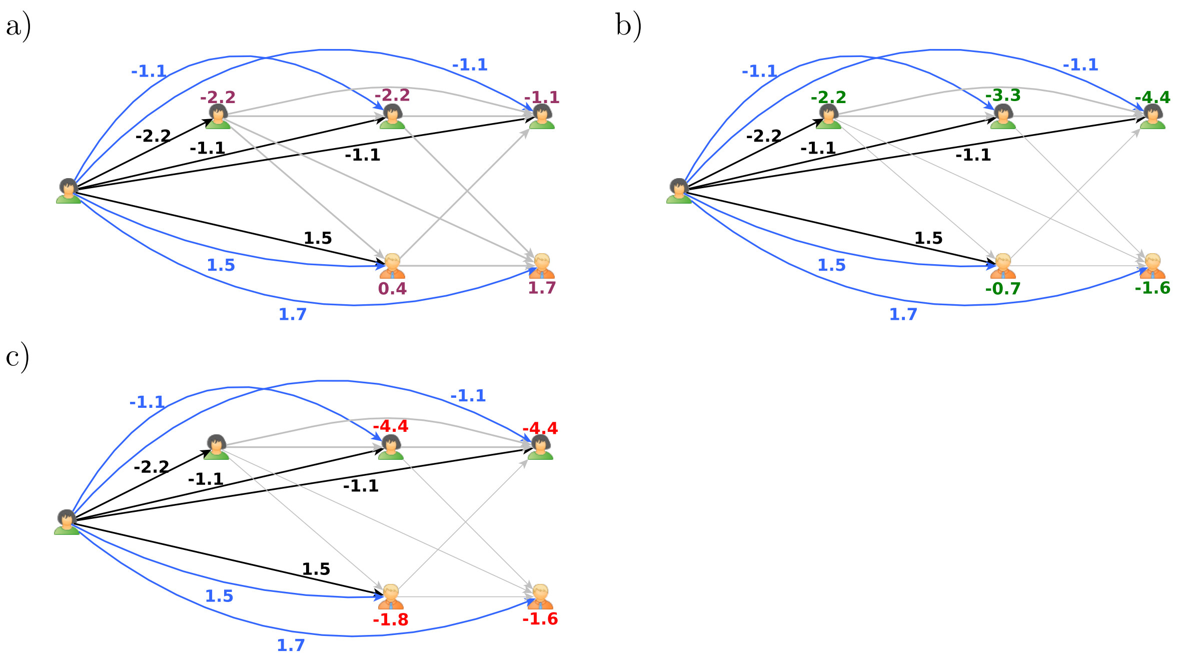


Figure 5.7: The principle of our subroutine used for obtaining lifted min marginals of an outflow subproblem (Algorithm MM-Not-Opt-Out). First, array lifted_cost must be obtained by running Algorithm Opt-Out-Cost (Figure a) as illustrated in Figure 5.3. Second, numbers of array back_cost are computed from left to right, eventually in the DFS manner (Algorithm Backward-DFS). They are depicted in green in (b). By adding the values from lifted_cost (violet in (a)) and back_cost (green in (b)) and subtracting the cost of the incident lifted edge, we obtain the constrained optimum where the respective lifted edge is active (red numbers in (c)). In case of computing all lifted min marginals at once (Algorithm All-Lifted-MM-Out), reparametrization and min marginals computation is performed at once. In that case, red numbers in (c) additionally incorporate reparametrization changes of their predecessors' costs. This is omitted here for clarity.

```
Algorithm Separation-Costs $(\theta^{in}, \theta^{out})$
Input Current cost in inflow and outflow factors $\theta^{in}, \theta^{out}$
Output Cost reparametrization $\forall uv \in E : \tilde{\theta}_{uv}$, $\forall uv \in E' : \tilde{\theta}'_{uv}$
 1: $\forall uv \in E : \tilde{\theta}_{uv} = 0$, $\forall uv \in E' : \tilde{\theta}'_{uv} = 0$
 2: for all $u \in V \setminus \{s, t\}$ do
 3:     $\forall uv \in \delta^+_E(u) : \gamma(u)^{out}_{uv} = 0$
 4:     $\forall uv \in \delta^-_E(u) : \gamma(u)^{in}_{vu} = 0$
 5:     $\gamma'(u)^{out} = 0.5 \cdot$All-Lifted-MM-Out$(u, \theta^{out(u)})$
 6:     $\gamma'(u)^{in} = 0.5 \cdot$All-Lifted-MM-In$(u, \theta^{in(u)})$
 7:     $\gamma(u)^{out} =$All-Base-MM-Out$(u, \theta^{out(u)} - (0, \gamma(u)^{out}, \gamma'(u)^{out}))$
 8:     $\gamma(u)^{in} =$All-Base-MM-In$(u, \theta^{in(u)} - (0, \gamma(u)^{in}, \gamma'(u)^{in}))$
 9:     $\forall uv \in \delta^+_E(u) : \tilde{\theta}_{uv} \mathrel{+}= \gamma(u)^{out}_{uv}$
10:     $\forall vu \in \delta^-_E(u) : \tilde{\theta}_{vu} \mathrel{+}= \gamma(u)^{in}_{vu}$
11:     $\forall uv \in \delta^+_{E'}(u) : \tilde{\theta}'_{uv} \mathrel{+}= \gamma'(u)^{out}_{uv}$
12:     $\forall vu \in \delta^-_{E'}(u) : \tilde{\theta}'_{vu} \mathrel{+}= \gamma'(u)^{in}_{vu}$
13: end for
```

```
Algorithm Connect $(i, j, \text{pred}, \text{desc}, E^1)$
Input $i, j, \text{pred}, \text{desc}, E^1$
 1: $E^1 \leftarrow ij$
 2: for all $p \in \text{pred}[i]$ do
 3:     for all $d \in \text{desc}[j]$ do
 4:         if $f_d - f_p \leq L_{\max}$ then
 5:             $\text{desc}[p] \leftarrow d$
 6:             $\text{pred}[d] \leftarrow p$
 7:         end if
 8:     end for
 9: end for
```

5.4.2 SEPARATION FOR PATH SUBPROBLEMS

Notation. Before we start with the description of the separation procedures, we define some symbols that are used in the algorithms in this and the following section.

- $G^1 = (V, E^1)$: a subgraph of base graph $G = (V, E)$ or multigraph $G \cup G' = (V, E \cup E')$ where the edge set E^1 is defined in the respective algorithms.
- $\text{pred}[v]$: Predecessors of vertex v (vertices from that v is reachable) in graph G^1.
- $\text{desc}[v]$: Descendants of vertex v (vertices that are reachable from v) in graph G^1.
- f_v : The number of the video frame where vertex v occurs.
- $L_{\max}$ the maximal distance between two video frames such that edges are created between their detections. We set $L_{\max} = 60$ or $L_{\max} = 50$ in our experiments.

Algorithm Separate-Path-Subproblem describes the path subproblem separation pro-

Algorithm Separate-Path-Subproblem (ε)

```
Input Cost threshold $\varepsilon$
1: $\tilde{\theta}$ =Separation-Costs$(\theta^{in}, \theta^{out})$
2: $G^1 = (V, E^1 = \emptyset)$
3: $E^- = \{vw \in E | \tilde{\theta}_{vw} < -\varepsilon\} \cup \{vw \in E' | \tilde{\theta}'_{vw} < -\varepsilon\}$
4: $E'^+ = \{vw \in E' | \tilde{\theta}'_{vw} > \varepsilon\}$
5: $\forall v \in V : \mathrm{desc}[v] = \{v\}, \mathrm{pred}[v] = \{v\}$
6: Priority-Queue $Q = \emptyset$
7: for all $ij \in E^-$ ascending in $\tilde{\theta}$ do
8:     if $ij \in E$ then $c_{ij} = \tilde{\theta}_{ij}$
9:     else $c_{ij} = \tilde{\theta}'_{ij}$
10:    if $j \notin \mathrm{desc}[i]$ then Inner-Paths$(i, j, c_{ij}, \mathrm{pred}, \mathrm{desc}, E^+, E^1, Q)$
11:    if $ij \in E'$ then Outer-Paths$(i, j, c_{ij}, \mathrm{pred}, \mathrm{desc}, E^+, E^1, Q)$
12:    Connect$(i, j, \mathrm{pred}, \mathrm{desc}, E^1)$
13: end for
```

Algorithm Inner-Paths $(i, j, c_{ij}, \mathrm{pred}, \mathrm{desc}, E^+, E^1, Q)$

```
Input $i, j, c_{ij}, \mathrm{pred}, \mathrm{desc}, E^+, E^1, Q$
1: for all $p \in \mathrm{pred}[i]$ do
2:     for all $d \in \mathrm{desc}[j]$ do
3:         if $pd \in E'^+ \wedge d \notin \mathrm{desc}[p]$ then
4:             $P_1$ =Find-Path$(p, i, E^1)$
5:             $P_2$ =Find-Path$(j, d, E^1)$
6:             $P = (P_1, ij, P_2)$
7:             $priority = \min\{|c_{ij}|, \tilde{\theta}'_{pd}\}$
8:             $Q \leftarrow (\text{Path-Problem}(P), priority)$
9:         end if
10:    end for
11: end for
```

cedure. The algorithm finds paths together with a lifted edge connecting the start and the end point of the path such that exactly one lifted edge has a positive cost, while all the remaining base and lifted edges have a negative cost.

First, lifted and base edge costs are obtained in Algorithm Separation-Costs by computing min-marginals of inflow and outflow factors. Second, a graph with an empty edge set E^1 is created. Then, edges with negative costs are added to E^1 in ascending order. After adding an edge, we check whether separating path subproblems with edge costs leading to lower bound improvement is possible. Such a factor must contain the newly added edge, one positive lifted edge, and edges that already are in the edge set E^1.

Algorithm Inner-Paths separates those paths subproblems where the only positive edge is the one connecting the path's endpoints. Algorithm Outer-Paths separates those path subproblems where the only positive edge is one of the edges within the path. Algorithm Connect updates connectivity structures by adding edge ij to the edge set E^1.

Algorithm Outer-Paths $(i, j, c_{ij}, \text{pred}, \text{desc}, E^+, E^1, Q)$

```
Input $i, j, c_{ij}, \text{pred}, \text{desc}, E^+, E^1, Q$
 1: for all $p \in \text{pred}[j]$ do
 2:     for all $d \in \text{desc}[i]$ do
 3:         if $dp \in E^+$ then
 4:             $P_1 =$Find-Path$(i, d, E^1)$
 5:             $P_2 =$Find-Path$(p, j, E^1)$
 6:             $P = (P_1, ij, P_2)$
 7:             $priority = \min\{|c_{ij}|, \tilde{\theta}'_{dp}\}$
 8:             $Q \leftarrow (\text{Path-Problem}(P), priority)$
 9:         end if
10:     end for
11: end for
```

It is sufficient to store the connectivity information only for those pairs of vertices that have time distance lower or equal to $L_{\max}$ which is the maximal frame distance used for creating edges.

Queue with subproblems. Each path subproblem has a guaranteed lower bound improvement, see Proposition 5.2. We add the found subproblems to priority queue Q where we sort w.r.t. the guaranteed lower bound improvement. After searching for new subproblems, we add the k best path and cut subproblems from queue Q to the optimization problem (set $\mathcal{S}$). Some variables typically occur in multiple subproblems. We allow to add maximally 4 path and cut subproblems that share a certain variable during one separation call.

5.4.3 SEPARATION FOR CUT SUBPROBLEMS

Algorithm Separate-Cut-Subproblem finds cuts consisting of base edges with positive costs and a lifted edge having endpoints on both sides of the cut and negative cost. Similarly as for the path subproblem separation, lifted and base edge costs are obtained by computing min-marginals of inflow and outflow subproblems in Algorithm Separation-Costs. Each edge $uv \in E'^-$ is a candidate lifted edge for a uv-cut subproblem.

The edge set E^1 initially contains all base edges with a cost lower than ε. The remaining base edges are added to E^1 in ascending order. Whenever a newly added edge ij causes a connection between u and v where $uv \in E'^-$, a uv-cut C is separated. We select cut C to contain only those edges that do not belong to E^1. This ensures that ij is the weakest cut edge. In the same time, C is the best uv-cut with respect to the cost of the weakest cut edge.

Similarly as for the paths, the found cut subproblems are added to a priority queue where the priority represents the guaranteed lower bound improvement (see Proposition 5.3) after adding the factor to our problem. We select from the queues containing the path and cut subproblems those to be added to our subproblems set $\mathcal{S}$.

Algorithm Separate-Cut-Subproblem (ε)

```
Input Cost threshold $\varepsilon$
1: $\tilde{\theta}$ =Separation-Costs$(\theta^{in}, \theta^{out})$
2: $E^- = \{vw \in E | \tilde{\theta}'_{vw} < \varepsilon\}$, $E^+ = E \setminus E^-$
3: $G^1 = (V, E^1 = \emptyset)$
4: for all $ij \in E^-$ do
5:     Connect$(i, j, \mathrm{pred}, \mathrm{desc}, E^1)$
6: end for
7: $E'^- = \{vw \in E' | \tilde{\theta}'_{vw} < -\varepsilon \wedge w \notin \mathrm{desc}[v]\}$
8: Priority-Queue $Q = \emptyset$
9: for all $ij \in E^+$ ascending in $\tilde{\theta}$ do
10:     for all $u \in \mathrm{pred}[i]$ do
11:         for all $v \in \mathrm{desc}[j]$ do
12:             if $uv \in E'^-$ then
13:                 $C =$ cut between $u, v$ using edges $E \setminus E^1$
14:                 $priority = \min\{\tilde{\theta}_{ij}, |\tilde{\theta}'_{uv}|\}$
15:                 $Q \leftarrow$ (Cut-Problem$(C, u, v)$, $priority$)
16:                 Delete $uv$ from $E'^-$
17:             end if
18:         end for
19:     end for
20:     Connect$(i, j, \mathrm{pred}, \mathrm{desc}, E^1)$
21: end for
```

5.4.4 TIGHTENING LOWER BOUND IMPROVEMENT

In order to show that the separation procedures in Algorithms Separate-Path-Subproblem and Separate-Cut-Subproblem lead to relaxations that improve the lower bound we show the following: (i) Certain reparametrization used in the above algorithms are non-decreasing in the lower bound. (ii) Separation procedures find new subproblems such that w.r.t. the above reparametrization, a guaranteed lower bound improvement can be achieved.

Points (i) and (ii) guarantee that the same lower bound achievement w.r.t. the original reparametrization can be found. The special reparametrization chosen helps empirically to find good subproblems.

Lemma 5.1. *Let $\mathsf{s} \in \mathcal{S}$ be a subproblem, θ its cost and $L_{\mathsf{s}}(\theta)$ its lower bound for cost θ. Given a cost reparametrization γ such that*

1. $\forall i \in [\mathsf{d}(\mathsf{s})]$

$$\gamma_i \begin{cases} \leq 0 & \text{if } \exists x^* \in \operatorname{argmin}_{x \in \mathcal{X}^{\mathsf{s}}} \langle \theta, x \rangle : x_i^* = 1 \\ \geq 0 & \text{if } \exists x^* \in \operatorname{argmin}_{x \in \mathcal{X}^{\mathsf{s}}} \langle \theta, x \rangle : x_i^* = 0 \end{cases} \tag{5.10}$$

2. $\operatorname*{argmin}_{x \in \mathcal{X}^{\mathsf{s}}} \langle \theta, x \rangle \subseteq \operatorname*{argmin}_{x \in \mathcal{X}^{\mathsf{s}}} \langle \theta - \gamma, x \rangle$

and a coordinate-wise scaled reparametrization $\gamma(\omega)$ defined by coefficients $\omega \in [0,1]^{\mathsf{d(s)}}$ where $\forall i \in [\mathsf{d(s)}] : \gamma(\omega)_i = \omega_i \gamma_i$, it holds:

1. *The lower bound of* s *after reparametrization $\gamma(\omega)$ is $L_\mathsf{s}(\theta - \gamma(\omega)) = L_\mathsf{s}(\theta) - \sum_{i\in[\mathsf{d(s)}]:\gamma_i<0} \omega_i\gamma_i$.*
2. $\operatorname*{argmin}_{x\in\mathcal{X}^\mathsf{s}} \langle\theta, x\rangle \subseteq \operatorname*{argmin}_{x\in\mathcal{X}^\mathsf{s}} \langle\theta - \gamma(\omega), x\rangle$

Proof. Due to Formula (5.10), the following holds:

$$(\exists x^*, x^{**} \in \operatorname*{argmin}_{x\in\mathcal{X}^\mathsf{s}} \langle\theta, x\rangle \text{ s.t. } x_i^* \neq x_i^{**}) \Rightarrow \gamma_i = 0\,. \tag{5.11}$$

First, we evaluate $\langle\theta - \gamma(\omega), x^*\rangle$ where $x^* \in \operatorname{argmin}_{x\in\mathcal{X}^\mathsf{s}}\langle\theta, x\rangle$.

$$\langle\theta - \gamma(\omega), x^*\rangle = \sum_{i\in[d_\mathsf{s}]} (\theta_i - \omega_i\gamma_i)x_i^* = \sum_{i\in[d_\mathsf{s}]} \theta_i x_i^* - \sum_{i\in[d_\mathsf{s}]:\gamma_i<0} \omega_i\gamma_i x_i^* = L_\mathsf{s}(\theta) - \sum_{i\in[d_\mathsf{s}]:\gamma_i<0} \omega_i\gamma_i \tag{5.12}$$

Second, we inspect $\langle\theta - \gamma(\omega), x\rangle$ for any $x \in \mathcal{X}^\mathsf{s}$.

$$\begin{aligned}
&\forall x \in \mathcal{X}^\mathsf{s}, \forall x^* \in \operatorname*{argmin}_{x\in X^\mathsf{s}}\langle\theta, x\rangle :\\
&\langle\theta - \gamma(\omega), x\rangle = \sum_{i\in[\mathsf{d(s)}]} (\theta_i - \omega_i\gamma_i)x_i = \sum_{i\in[\mathsf{d(s)}]} (\theta_i - \gamma_i)x_i + \sum_{i\in[\mathsf{d(s)}]} (1-\omega_i)\gamma_i x_i \geq\\
&\qquad \geq L_\mathsf{s}(\theta - \gamma) + \sum_{i\in[\mathsf{d(s)}]:\gamma_i<0} (1-\omega_i)\gamma_i x_i \geq\\
&\qquad \geq L_\mathsf{s}(\theta - \gamma) + \sum_{i\in[\mathsf{d(s)}]:\gamma_i<0} (1-\omega_i)\gamma_i =\\
&\qquad = \sum_{i\in[\mathsf{d(s)}]} (\theta_i - \gamma_i)x_i^* + \sum_{i\in[\mathsf{d(s)}]} (1-\omega_i)\gamma_i x_i^* = \sum_{i\in[\mathsf{d(s)}]} (\theta_i - \omega_i\gamma_i)x_i^* =\\
&\qquad = \langle\theta - \gamma(\omega), x^*\rangle
\end{aligned} \tag{5.13}$$

Formula (5.13) proves Point 2 of Lemma 5.1. Formulas (5.12) and (5.13) together prove Point 1. □

Lemma 5.2. *Variables $(\gamma(u)^{out}, \gamma'(u)^{out})$, resp. $(\gamma(u)^{in}, \gamma'(u)^{in})$ in Algorithm* Separation-Costs *satisfy the requirements of Lemma 5.1 for each outflow resp. inflow subproblem of vertex u.*

Proof. Both Algorithms All-Base-MM-Out and All-Lifted-MM-Out output reparametrization variables that satisfy the requirements of Lemma 5.1. We have, for an outflow subproblem of node u:

$$\begin{aligned}
\operatorname*{argmin}_{(z,y,y')\in\mathcal{X}^{out(u)}} \langle\theta, (z,y,y')\rangle &\subseteq \operatorname*{argmin}_{(z,y,y')\in\mathcal{X}^{out(u)}} \langle\theta - (\mathbb{0}, \gamma'(u)^{out}), (z,y,y')\rangle\\
&\subseteq \operatorname*{argmin}_{(z,y,y')\in\mathcal{X}^{out(u)}} \langle\theta - (0, \gamma(u)^{out}, \gamma'(u)^{out}), (z,y,y')\rangle
\end{aligned} \tag{5.14}$$

Therefore, also $(\gamma(u)^{out}, \gamma'(u)^{out})$ together satisfy the requirements of Lemma 5.1. Analogically, for the inflow subproblems. □

Costs in the new path and cut subproblems. One edge is typically shared among multiple newly added path and cut subproblems. Therefore, the available cost reparametrizations $\tilde{\theta}$ and $\tilde{\theta}'$ from Algorithm Separation-Costs must be redistributed to the newly added subproblems. We denote the set of all newly added path subproblems resp. cut subproblems in tightening iteration i by $\mathcal{P}^i$ resp. $\mathcal{C}^i$. For each base resp. lifted edge uv, we sum up the number of newly added path and cut subproblems that contain uv.

$$\begin{aligned} N_{uv} =& |\{P \in \mathcal{P}^i | uv \in P_E\}| + |\{C \in \mathcal{C}^i | uv \in C_E\}|, \\ N'_{uv} =& |\{P \in \mathcal{P}^i | uv \in P_{E'}\}| + |\{P \in \mathcal{P}^i | P \text{ is a } uv\text{-path}\}| + |\{C \in \mathcal{C}^i | C \text{ is a } uv\text{-cut}\}| . \end{aligned} \tag{5.15}$$

Then, we define coefficient ω_{uv} resp. ω'_{uv} for each base edge $uv \in E$ resp. lifted edge $uv \in E'$ that belongs to a newly added subproblem as

$$\omega_{uv} = \frac{1}{N_{uv}}, \quad \omega'_{uv} = \frac{1}{N'_{uv}}. \tag{5.16}$$

Finally, for each newly added path subproblem P resp. cut subproblem C, we set the cost of base edge $uv \in E$ to $\theta^P_{uv} = \omega_{uv}\tilde{\theta}_{uv}$, resp. $\theta^C_{uv} = \omega_{uv}\tilde{\theta}_{uv}$. Analogically, for the lifted edges. As mentioned in Section 5.4.2, we add maximally 4 path and cut subproblems sharing one edge in one separation call. Therefore, N_{uv} resp. N'_{uv} are lower or equal to 4 for each base resp. lifted edge uv.

Cost update in in/outflow subproblems. If we use an edge uv for creating one or more path and cut subproblems, it is necessary to update its cost in the inflow subproblem of vertex v and the outflow subproblem of vertex u accordingly. For instance, we update the cost of base edge uv in the outflow subproblem of u as follows $\theta^{out}_{uv} -= \gamma(u)^{out}_{uv}$. Where we adopt the notation from Algorithm Separation-Costs. Note that $\tilde{\theta}_{uv} = \gamma(v)^{in}_{uv} + \gamma(u)^{out}_{uv}$. Therefore, the total cost of edge variable uv is preserved.

Proposition 5.2 (Guaranteed lower bound improvement of path subproblem)**.** *If a path subproblem corresponding to vw-path P separated by Algorithm* Separate-Path-Subproblem *is added to the subproblem set $\mathcal{S}$, the guaranteed improvement of the global lower bound is* $\min\{\min_{uv \in P_E} |\theta^P_{uv}|, \min_{uv \in P_{E'} \cup vw} |\theta'^P_{uv}|\}$*, where θ^P is the reparametrized cost used for the path factor initialization.*

Proof. We simplify the notation from Algorithm Separation-Costs such that we use γ^{out}_{uv} resp. γ^{in}_{uv} instead of $\gamma(u)^{out}_{uv}$ resp. $\gamma(v)^{in}_{uv}$ for better readability. Similarly for the dashed variables representing reparametrizations of lifted edges.

Algorithm Separate-Path-Subproblem separates only those subproblems that contain exactly one lifted edge with cost $\theta'^P_{uv} > \varepsilon$ and the rest of the edges have cost lower than $-\varepsilon$. The reparametrized costs of the path factor are fractions of cost reparametrizations obtained by Algorithm Separation-Costs. We have

$$\begin{aligned} \forall uv \in P_E : \quad & \theta^P_{uv} = \omega_{uv} \cdot (\gamma^{out}_{uv} + \gamma^{in}_{uv}), \\ & \theta^{out}_{uv} -= \omega_{uv} \cdot \gamma^{out}_{uv}, \qquad \theta^{in}_{uv} -= \omega_{uv} \cdot \gamma^{in}_{uv}, \\ \forall uv \in P_{E'} : \quad & \theta'^P_{uv} = \omega'_{uv} \cdot (\gamma'^{out}_{uv} + \gamma'^{in}_{uv}), \\ & \theta'^{out}_{uv} -= \omega'_{uv} \cdot \gamma'^{out}_{uv}, \qquad \theta'^{in}_{uv} -= \omega'_{uv} \cdot \gamma'^{in}_{uv} \end{aligned} \tag{5.17}$$

We evaluate the change of the lower bounds of all relevant inflow and outflow factors after reparametrization given by Formula (5.17). According to Lemma 5.1, we have

$$\begin{aligned}
\Delta L^{out} + \Delta L^{in} = \\
&= -\sum_{uv \in P_E : \gamma^{out}_{uv} < 0} \omega_{uv} \cdot \gamma^{out}_{uv} - \sum_{uv \in P_{E'} \cup vw : \gamma'^{out}_{uv} < 0} \omega'_{uv} \cdot \gamma'^{out}_{uv} \\
&\quad - \sum_{uv \in P_E : \gamma^{in}_{uv} < 0} \omega_{uv} \cdot \gamma^{in}_{uv} - \sum_{uv \in P_{E'} \cup vw : \gamma'^{in}_{uv} < 0} \omega'_{uv} \cdot \gamma'^{in}_{uv} \geq \qquad (5.18) \\
&\geq -\sum_{uv \in P_E : \gamma^{out}_{uv} + \gamma^{in}_{uv} < 0} \omega_{uv} \cdot (\gamma^{out}_{uv} + \gamma^{in}_{uv}) - \sum_{uv \in P_{E'} \cup vw : \gamma'^{in}_{uv} + \gamma'^{out}_{uv} < 0} \omega'_{uv} \cdot (\gamma'^{out}_{uv} + \gamma'^{in}_{uv}) = \\
&= -\sum_{uv \in P_E : \theta^P_{uv} < 0} \theta^P_{uv} - \sum_{uv \in P_{E'} \cup vw : \theta'^P_{uv} < 0} \theta'^P_{uv}
\end{aligned}$$

Let $kl \in P_{E'}$ be the only lifted edge with positive cost in the path subproblem. We set $\alpha = \min\{\min_{ij \in P_E} |\theta^P_{ij}|, \min_{ij \in P_{E'} \cup vw \setminus kl} |\theta'^P_{ij}|\}$ as in Algorithm Path-Subproblem-Optimization. If we denote by ΔL^P the lower bound of the newly separated path subproblem, the global lower bound change after adding the path subproblem is:

$$\Delta L = \Delta L^{out} + \Delta L^{in} + \Delta L^P \qquad (5.19)$$

If $\alpha < \theta'^P_{kl}$:

$$\begin{aligned}
\Delta L^{out} + \Delta L^{in} + \Delta L^P \geq &-\sum_{uv \in P_E : \theta^P_{uv} < 0} \theta^P_{uv} - \sum_{uv \in P_{E'} : \theta'^P_{uv} < 0} \theta'^P_{uv} + \qquad (5.20) \\
&+ \sum_{uv \in P_E : \theta^P_{uv} < 0} \theta^P_{uv} + \sum_{uv \in P_{E'} : \theta'^P_{uv} < 0} \theta'^P_{uv} + \alpha = \alpha
\end{aligned}$$

If $\alpha \geq \theta'^P_{kl}$:

$$\begin{aligned}
\Delta L^{out} + \Delta L^{in} + \Delta L^P \geq &-\sum_{uv \in P_E : \theta^P_{uv} < 0} \theta^P_{uv} - \sum_{uv \in P_{E'} \cup vw : \theta'^P_{uv} < 0} \theta'^P_{uv} + \qquad (5.21) \\
&+ \sum_{uv \in P_E : \theta^P_{uv} < 0} \theta^P_{uv} + \sum_{uv \in P_{E'} : \theta'^P_{uv}} \theta'^P_{uv} = \theta'^P_{kl}
\end{aligned}$$

□

Proposition 5.3 (Guaranteed lower bound improvement of cut subproblem)**.** *If a subproblem corresponding to vw-cut C separated by Algorithm* Separate-Cut-Subproblem *is added to the subproblem set $\mathcal{S}$, the guaranteed improvement of the global lower bound is* $\min\{\min_{uv \in C} \theta^C_{uv}, |\theta'^C_{vw}|\}$. *Where θ^C is the reparametrized cost used for the cut factor initialization.*

Proof. We obtain the reparametrized cost θ^C for the cut subproblem analogically as in Formula (5.17) for the path subproblem. Note that Algorithm Separate-Cut-Subproblem ensures that all cut edges in the separated cut subproblem have positive cost and the lifted edge vw has a negative cost. Using the same arguments as in the proof of Proposition 5.3, we obtain the lower bound change of inflow and outflow factors after separating the cut subproblem:

$$\begin{aligned}\Delta L^{out} + \Delta L^{in} &= -\sum_{uv\in C:\gamma^{out}_{uv}<0} \omega_{uv}\cdot\gamma^{out}_{uv} - \sum_{uv\in C:\gamma^{in}_{uv}<0} \omega_{uv}\cdot\gamma^{in}_{uv} - \omega'_{vw}\gamma'^{out}_{vw} - \omega'_{vw}\gamma'^{in}_{vw} \geq \\ &\geq -\sum_{uv\in C:\gamma^{out}_{uv}+\gamma^{in}_{uv}<0} \omega_{uv}\cdot(\gamma^{out}_{uv}+\gamma^{in}_{uv}) - \omega'_{vw}\gamma'^{out}_{vw} - \omega'_{vw}\gamma'^{in}_{vw} = \\ &= -\,\omega'_{vw}\gamma'^{out}_{vw} - \omega'_{vw}\gamma'^{in}_{vw} = -\theta'^{C}_{vw}\end{aligned} \tag{5.22}$$

Algorithm Cut-Subproblem-Optimization shows how we obtain the lower bound of the cut subproblem. We set $\theta^C_{ij} = \operatorname{argmin}_{uv\in C}\theta^C_{uv}$. If $\theta^C_{ij} < |\theta'^C_{vw}|$, we get the overall lower bound improvement

$$\Delta L^{out} + \Delta L^{in} + \Delta L^{C} \geq -\theta'^C_{vw} + \theta'^C_{vw} + \theta^C_{ij} = \theta^C_{ij}\,. \tag{5.23}$$

If $\theta^C_{ij} \geq |\theta'^C_{vw}|$, the lower bound of the cut subproblem is 0 and the overall lower bound improvement is

$$\Delta L^{out} + \Delta L^{in} + \Delta L^{C} \geq -\theta'^C_{vw}\,. \tag{5.24}$$

$\square$

5.4.5 MESSAGE PASSING

One solver run consists of subproblems initialization and a number of message passing iterations. Algorithm Message-Passing details the whole run. Algorithms Inflow-Subproblem-Message-Passing, Path-Subproblem-Message-Passing, Cut-Subproblem-Message-Passing and Lower-Bound present methods that are called within one iteration.

The number of iterations is predetermined by an input parameter. We use typically tens or maximally one hundred iterations in our experiments.

Algorithm Message-Passing sends in each iteration messages between all subproblems in the subproblem set $\mathcal{S}$. Each subproblem creates messages to be sent by computing min-marginals of its variables (see Formula (5.2)). These min-marginals are re-scaled and redistributed between other subproblems that contain the respective variables. These operations are called reparametrizations. See Section 5.3 for details.

Algorithm Inflow-Subproblem-Message-Passing shows sending messages from the inflow subproblem of node u. Algorithm Path-Subproblem-Message-Passing shows sending messages from a path subproblem. Algorithm Cut-Subproblem-Message-Passing presents sending messages from a cut subproblem.

Algorithm Lower-Bound computes a lower bound of the (LDP) objective by summing up the lower bounds of all subproblems. The cost reparametrization realized via our message passing procedures ensures that the lower bound is non-decreasing during the computation.

Algorithm Message-Passing (G, G', θ)

Input Graphs $G = (V, E)$ and $G' = (V, E')$, costs $\theta \in \mathbb{R}^{V \cup E \cup E'}$
Output Best found primal solution $(z, y, y')^{\mathsf{ub}}$, lower bound LB

```
1: **Initialization:**
2: for $v \in V$ do
3:     Add inflow subproblem for node $v$.
4:     $$\forall uv \in \delta^-_E(v): \quad \theta^{in}_{uv} = \begin{cases} \theta_{uv} & \text{if } v = s \\ \frac{1}{2}\theta_{uv} & \text{otherwise} \end{cases}$$
5:     $\forall uv \in \delta^-_{E'}(v)$: $\theta'^{in}_{uv} = \frac{1}{2}\theta'_{uv}$.
6:     $\theta^{in}_v = \frac{1}{2}\theta_v$.
7:     Add outflow subproblem for node $v$ with analoguous costs.
8: end for
9: $\mathcal{C} = \emptyset$
10: $\mathcal{P} = \emptyset$
11: **Lagrange decomposition optimization**
12: for iter $= 1, \dots,$ max_iter do
13:     **Forward Pass:**
14:     for $u = u_1, \dots, u_{|V|}$ do
15:         Inflow-Subproblem-Message-Passing$(u)$
16:         Outflow-Subproblem-Message-Passing$(u)$
17:     end for
18:     for $P \in \mathcal{P}$ do
19:         Path-Subproblem-Message-Passing$(P)$
20:     end for
21:     for $C \in \mathcal{C}$ do
22:         Cut-Subproblem-Message-Passing$(C)$
23:     end for
24:     **Backward Pass**:
25:     Revert order of nodes and perform above iteration.
26:     if iter $\equiv 0 \mod k$ then
27:         Separate-Cut-Subproblem$(\varepsilon)$
28:         Separate-Path-Subproblem$(\varepsilon)$
29:         Add cut and path subproblems to $\mathcal{C}$ and $\mathcal{P}$
30:     end if
31:     if iter $\equiv 0 \mod l$ then
32:         $(z, y, y') =$Compute-Primal$(\mathcal{S}, \theta)$
33:         if $\langle \theta, (z, y, y') \rangle < \langle \theta, (z, y, y')^{\mathsf{ub}} \rangle$ then
34:             $(z, y, y')^{\mathsf{ub}} = (z, y, y')$
35:         end if
36:     end if
37:     $LB =$Lower-Bound
38: end for
```

Algorithm Inflow-Subproblem-Message-Passing (u)

Input central vertex u of the subproblem

```
1: $\gamma'^{in}$ =All-Lifted-MM-In$(u, \theta^{in(u)})$.
2: $\omega = 1$
3: for $vu \in \delta^-_E(u), P \in \mathcal{P} : vu \in P_E$ do
4:     $\gamma^P_{vu}$ = Path-Base-Min-Marginal$(u, v, \theta^P)$
5:     $\omega$ += 1
6: end for
7: for $vu \in \delta^-_{E'}(u), P \in \mathcal{P} : vu \in P_{E'}$ do
8:     $\gamma'^P_{vu}$ = Path-Lifted-Min-Marginal$(u, v, \theta^P)$
9:     $\omega$ += 1
10: end for
11: for $vu \in \delta^-_{E'}(u), P \in \mathcal{P} : P \in vu$-paths$(G)$ do
12:     $\gamma'^P_{vu}$ = Path-Lifted-Min-Marginal$(u, v, \theta^P)$
13:     $\omega$ += 1
14: end for
15: for $vu \in \delta^-_E(u), C \in \mathcal{C} : vu \in C_E$ do
16:     $\gamma^C_{vu}$ =Cut-Base-Min-Marginal$(u, v, \theta^C)$
17:     $\omega$ += 1
18: end for
19: for $vu \in \delta^-_{E'}(u), C \in \mathcal{C} : C$ is a $vu$-Cut do
20:     $\gamma'^C_{vu}$ = Cut-Lifted-Min-Marginal$(u, v, \theta^C)$
21:     $\omega$ += 1
22: end for
23: for $vu \in \delta^-_{E'}(u)$ do
24:     $\theta^{in}_{vu}$ −= $\frac{1}{\omega} \cdot \gamma'^{in}_{vu}$,   $\theta^{out}_{vu}$ += $\frac{1}{\omega} \cdot \gamma'^{in}_{vu}$
25: end for
26: for $vu \in \delta^-_E(u), P \in \mathcal{P} : vu \in P_E$ do
27:     $\theta^{in}_{vu}$ −= $\frac{1}{\omega} \cdot \gamma^P_{vu}$,   $\theta^P_{vu}$ += $\frac{1}{\omega} \cdot \gamma^P_{vu}$
28: end for
29: for $vu \in \delta^-_{E'}(u), P \in \mathcal{P} : vu \in P_{E'}$ do
30:     $\theta'^{in}_{vu}$ −= $\frac{1}{\omega} \cdot \gamma'^P_{vu}$,   $\theta'^P_{vu}$ += $\frac{1}{\omega} \cdot \gamma'^P_{vu}$
31: end for
32: for $vu \in \delta^-_{E'}(u), P \in \mathcal{P} : P \in vu$-paths$(G)$ do
33:     $\theta'^{in}_{vu}$ −= $\frac{1}{\omega} \cdot \gamma'^P_{vu}$,   $\theta'^P_{vu}$ += $\frac{1}{\omega} \cdot \gamma'^P_{vu}$
34: end for
35: for $vu \in \delta^-_E(u), C \in \mathcal{C} : vu \in C_E$ do
36:     $\theta^{in}_{vu}$ −= $\frac{1}{\omega} \cdot \gamma^C_{vu}$,   $\theta^C_{vu}$ += $\frac{1}{\omega} \cdot \gamma^C_{vu}$
37: end for
38: for $vu \in \delta^-_{E'}(u), C \in \mathcal{C} : C$ is a $vu$-Cut do
39:     $\theta'^{in}_{vu}$ −= $\frac{1}{\omega} \cdot \gamma'^C_{vu}$,   $\theta'^C_{vu}$ += $\frac{1}{\omega} \cdot \gamma'^C_{vu}$
40: end for
```

Algorithm Path-Subproblem-Message-Passing (P)

Input: uv-Path $P \in \mathcal{P}$

```
1: $\gamma^P =$ Path-Min-Marginals$(P, \theta^P)$
2: $\omega^P = \frac{1}{2 \cdot |P_E| + 2 \cdot |P_{E'}|}$
3: for $kl \in P_E$ do
4:     $\theta^P_{kl} -= 2\omega^P \cdot \gamma^P_{kl}$
5:     $\theta^{in}_{kl} -= \omega^P \cdot \gamma^P_{kl}$,   $\theta^{out}_{kl} -= \omega^P \cdot \gamma^P_{kl}$
6: end for
7: for $kl \in P_{E'}$ do
8:     $\theta^P_{kl} -= 2\omega^P \cdot \gamma^P_{kl}$
9:     $\theta^{in}_{kl} -= \omega^P \cdot \gamma^P_{kl}$,   $\theta^{out}_{kl} -= \omega^P \cdot \gamma^P_{kl}$
10: end for
```

Algorithm Cut-Subproblem-Message-Passing (C)

Input: uv-Cut $C \in \mathcal{C}$

```
1: $\gamma^C =$Cut-Min-Marginals$(C, \theta^C)$
2: $\omega^C = \frac{1}{2 \cdot |C_E| + 2}$
3: for $kl \in C_E$ do
4:     $\theta^C_{kl} -= 2\omega^C \cdot \gamma^C_{kl}$
5:     $\theta^{in}_{kl} += \omega^C \cdot \gamma^C_{kl}$,   $\theta^{out}_{kl} += \omega^C \cdot \gamma^C_{kl}$
6: end for
7: $\theta'^C_{uv} -= 2\omega^C \cdot \gamma'^C_{uv}$
8: $\theta^{in}_{uv} += \omega^C \cdot \gamma'^C_{uv}$,   $\theta^{out}_{uv} += \omega^C \cdot \gamma'^C_{uv}$
```

Algorithm Lower-Bound $(\mathcal{S})$

Input Subproblems $\mathcal{S}$

Output Lower bound value LB

```
1: $LB = 0$
2: for $u \in V \backslash \{s, t\}$ do
3:     $LB +=$Opt-In-cost$(u, \theta^{in})$
4:     $LB +=$Opt-Out-cost$(u, \theta^{out})$
5: end for
6: for $P \in \mathcal{P}$ do
7:     $LB +=$ Path-Subproblem-Optimization$(P, \theta^P)$
8: end for
9: for $C \in \mathcal{C}$ do
10:     $LB +=$ Cut-Subproblem-Optimization$(C, \theta^C)$
11: end for
```

5.4.6 PRIMAL SOLUTION AND LOCAL SEARCH

Algorithm Compute-Primal in Section 5.3.6 summarizes the whole procedure for obtaining a primal solution. As stated in that section, we obtain an initial primal solution by solving the (MCF) problem.

Given a feasible solution of (LDP), Algorithm Local-Search improves it by splitting and merging paths. While we obtain the costs for (MCF) from the base and lifted edges costs in inflow and outflow factors (Algorithm Init-MCF), the local search procedure uses the original input costs of base and lifted edges.

Algorithm Check-Path-Split finds candidate split point of each path and recursively splits the path if the split leads to a decrease of the objective function.

For each vertex of each path, function *split* evaluates the cost of splitting the path after the vertex:

$$\forall v_j \in P_V = (v_1, \dots v_n): \quad split(v_j, P) = -\sum_{\substack{k \le j, l > j, \\ v_k v_l \in E'}} \theta'_{v_k v_l} - \theta_{v_j v_{j+1}} + \theta_{s v_{j+1}} + \theta_{v_j t} \tag{5.25}$$

The second step of the primal solution post-processing by Algorithm Local-Search is merging paths. Before the path merging itself, some candidate pairs of paths need to be shortened at their ends in order to enable their feasible merging.

Algorithm Shorten-For-Merge identifies pairs of those paths whose merging should lead to objective improvement but that cannot be connected directly due to missing base edge between their endpoints. In order to identify the desired paths pairs, several functions are used.

Function $l^+(P_1, P_2)$ resp $l^-(P_1, P_2)$ is the sum of positive resp. negative lifted edges from path P_1 to path P_2. Function $l(P_1, P_2)$ sums all lifted edges from P_1 to P_2.

$$\begin{aligned}
&\forall P_1, P_2 \in \mathcal{P}: \\
&\qquad l^+(P_1, P_2) = \sum_{uv \in E': u \in P_1, v \in P_2, \theta'_{uv} \ge 0} \theta'_{uv} \\
&\qquad l^-(P_1, P_2) = \sum_{uv \in E': u \in P_1, v \in P_2, \theta'_{uv} < 0} \theta'_{uv} \\
&\qquad l(P_1, P_2) = l^+(P_1, P_2) + l^-(P_1, P_2)
\end{aligned} \tag{5.26}$$

We use the above values in functions *merge* and *merge*$_\tau$ that evaluate the gain of merging two paths. Threshold $\tau \le 1$ constraints the ratio between the positive and the negative part of lifted cost function l that is considered acceptable for merging two paths.

$$\begin{aligned}
&\forall P_1 = (v_1, \dots, v_n), P_2 = (u_1, \dots, u_m) \in \mathcal{P}: \\
&\qquad merge(P_1, P_2) = \begin{cases} \theta_{v_n u_1} + l(P_1, P_2) & \text{if } v_n u_1 \in E \\ \infty & \text{otherwise} \end{cases}
\end{aligned} \tag{5.27}$$

Algorithm Local-Search (z, y, y')

Input Input primal solution z, y, y'

Output Improved primal solution z, y, y'

```
1: Obtain set of disjoint paths $\mathcal{P} = \{P_1, \dots, P_n\}$ from $y$
2: for all $P \in \mathcal{P}$ do
3:     $\mathcal{P}$ =Check-Path-Split$(P_i, \mathcal{P})$
4: end for
5: $\mathcal{P}$ =Shorten-For-Merge$(\mathcal{P})$
6: while $true$ do
7:     $(P_1, P_2) = \operatorname{argmin}_{(P_i,P_j)\in\mathcal{P}\times\mathcal{P}} merge_\tau(P_i, P_j) - out(P_i) - in(P_j)$
8:     if $merge_\tau(P_1, P_2) - out(P_1) - in(P_2) < 0$ then
9:         $\mathcal{P}$ =Merge-Paths$(P_1, P_2, \mathcal{P})$
10:    else
11:        break
12:    end if
13: end while
14: $(z, y, y')$=Set-From-Paths$(\mathcal{P})$
```

$$\forall P_1 = (v_1, \dots, v_n), P_2 = (u_1, \dots, u_m) \in \mathcal{P} :$$
$$merge_\tau(P_1, P_2) = \begin{cases} \infty & \text{if } v_n u_1 \notin E \vee \\ & \quad l^+(P_1, P_2) > \tau |l^-(P_1, P_2)| \\ \theta_{v_n u_1} + l(P_1, P_2) & \text{otherwise} \end{cases} \tag{5.28}$$

Algorithm Cut-Ends is applied on all paths pairs found by Algorithm Shorten-For-Merge. It inspects whether shortening of one or both paths leads to a feasible connection that ensures a desired objective improvement. It iteratively removes either the last vertex of the first path or the first vertex of the second path and checks if a connection is possible and how much it costs.

The last part of Algorithm Local-Search considers merging paths. We use formula $merge_\tau(P_i, P_j) - out(P_i) - in(P_j)$ to evaluate whether merging two paths is beneficial. Here $in(P_j)$ denotes input cost to the first vertex of P_j and $out(P_i)$ denotes output cost from the last vertex of P_i. We state the full formula just for completeness. We set the input and the output costs to zeros in our experiments. Using $merge_\tau$ ensures that we connect the paths only if the ratio between the positive lifted cost l^+ and negative lifted cost l^- between the paths is below the acceptable threshold.

Algorithm Check-Path-Split (P)

Input Input path P, set of all paths $\mathcal{P}$

Output Set of paths $\mathcal{P}$

```
1: $v_m = \text{argmax}_{v_j \in P_V} \, split(v_j, P)$
2: if $split(v_m, P) < 0$ then
3:     $(P_1, P_2) =$ Split-Path$(P, v_m)$
4:     $\mathcal{P}$.remove$(P)$, $\mathcal{P}$.insert$(P_1)$, $\mathcal{P}$.insert$(P_2)$
5:     $\mathcal{P} =$ Check-Path-Split$(P_1, \mathcal{P})$
6:     $\mathcal{P} =$ Check-Path-Split$(P_2, \mathcal{P})$
7: end if
8: return $\mathcal{P}$
```

Algorithm Shorten-For-Merge ($\mathcal{P}$)

Input Set of paths $\mathcal{P}$

Output Updated set of paths $\mathcal{P}$

```
1: for all $P_1 = (v_1, \dots, v_n) \in \mathcal{P}$ do
2:     $P = \underset{P_2=(u_1,\dots,u_m)\in\mathcal{P}: v_n u_1 \in E}{\text{argmin}} merge(P_1, P_2)$
3:     $P' = \underset{P_2=(u_1,\dots,u_m)\in\mathcal{P}: v_n u_1 \notin E}{\text{argmin}} l(P_1, P_2)$
4:     if $l(P_1, P') < merge(P_1, P) \wedge l(P_1, P') < 0$ then
5:         $P^* = P'$, $c = l(P_1, P')$
6:     else
7:         $P^* = P$, $c = merge(P_1, P)$
8:     end if
9:     if pred$[P^*] = \emptyset \vee$ score$[P^*] > c$ then
10:        pred$[P^*] = P_1$, score$[P^*] = c$
11:    end if
12: end for
13: for all $P_2 = (u_1, \dots, u_m) \in \mathcal{P}$ do
14:     if pred$[P_2]$=$P_1 = (v_1, \dots, v_n) \wedge v_n u_1 \notin E$ then
15:         $\mathcal{P} =$ Cut-Ends$(P_1, P_2, \mathcal{P})$
16:     end if
17: end for
```

Algorithm Cut-Ends $(P_1, P_2, \mathcal{P}, i_{max})$

Input $P_1 = (v_1, \dots, v_m), P_2 = (u_1, \dots, u_m)$, $\mathcal{P}$, maximum number of vertices to cut i_{max}

Output New set of paths $\mathcal{P}$

```
1: $c_1 = \infty, c_2 = \infty, i_1 = 0, i_2 = 0$
2: **while** $i_1 + i_2 < i_{max}$ **do**
3:     $P_1' = (v_1, \dots, v_{n-i_1})$, $P_2' = (u_{1+i_2}, \dots, u_m)$
4:     $P_1'' = (v_1, \dots, v_{n-i_1-1})$, $P_2'' = (u_{2+i_2}, \dots, u_m)$
5:     **if** $merge(P_1', P_2'') + merge(P_1'', P_2') < \infty$ **then**
6:         $\alpha_1 = merge(P_1', P_2'') + split(P_1, v_{n-i_1}) + split(P_2, u_{1+i_2})$
7:         $\alpha_2 = merge(P_1'', P_2') + split(P_1, v_{n-i_1-1}) + split(P_2, u_{i_2})$
8:         **if** $\alpha_1 < \alpha_2$ **then** $c_1 = i_1, c_2 = i_2 + 1$
9:         **else** $c_1 = i_1 + 1, c_2 = i_2$
10:        break
11:    **else if** $merge(P_1', P_2'') < \infty$ **then**
12:        $c_1 = i_1, c_2 = i_2 + 1$
13:        break
14:    **else if** $merge(P_1'', P_2') < \infty$ **then**
15:        $c_1 = i_1 + 1, c_2 = i_2$
16:        break
17:    **else**
18:        $\alpha_1 = l(P_1', P_2'') + split(P_1, v_{n-i_1}) + split(P_2, u_{1+i_2})$
19:        $\alpha_2 = l(P_1'', P_2') + split(P_1, v_{n-i_1-1}) + split(P_2, u_{i_2})$
20:        **if** $\alpha_1 < \alpha_2$ **then** $i_2$ ++
21:        **else** $i_1$ ++
22:    **end if**
23: **end while**
24: **if** $c_1 \neq \infty \wedge c_2 \neq \infty$ **then**
25:    $P_1' = (v_1, \dots, v_{n-c_1})$, $P_2' = (u_{1+c_2}, \dots, u_m)$
26:    **if** $merge_\tau(P_1', P_2') < 0$ **then**
27:        **if** $c_1 > 0$ **then**
28:            $(P_{11}, P_{12})$ =Split-Path$(P_1, v_{n-c_1})$
29:            $\mathcal{P}$.remove$(P_1)$
30:            $\mathcal{P}$.insert$(P_{11})$, $\mathcal{P}$.insert$(P_{12})$
31:        **end if**
32:        **if** $c_2 > 0$ **then**
33:            $(P_{21}, P_{22})$ =Split-Path$(P_2, u_{c_2})$
34:            $\mathcal{P}$.remove$(P_2)$
35:            $\mathcal{P}$.insert$(P_{21})$, $\mathcal{P}$.insert$(P_{22})$
36:        **end if**
37:    **end if**
38: **end if**
39: return $\mathcal{P}$
```

5.5 COMPUTATIONAL COMPLEXITY

Notation. We start with providing some symbols that are needed for the complexity analysis. For other symbols that are not stated here please refer to the respective sections containing the algorithms or to the list of symbols.

- K : The maximal number of base edges connecting a vertex with vertices in a certain time frame.
- D : The maximal number of vertices in one time frame.

We need $\mathcal{O}(|E^{inp}|)$ space where E^{inp} are all edges before graph sparsification.

The solver terminates if one of the following conditions is satisfied. Either the lower bound is equal to the objective value of the best primal solution, i.e. optimum has been found. Or the maximum number of message passing iterations has been reached. The optimum was not found in our experiments, so the latter condition applied.

The runtime of the solver is, therefore, determined by the input parameter denoting the maximum number of iterations. The dependence on the number of iterations is not exactly linear because the problem size grows with the number of path and cut subproblems added to set of subproblems $\mathcal{S}$ via cutting plane separation (see Sections 5.4.2 and 5.4.3).

An overview of the whole solver run and the tasks performed within one its iteration is given in Section 5.4.5. The runtime of the tasks is given by the runtime of computing min-marginals of the subproblems and by the separation of the new subproblems.

We discuss the complexity of the used algorithms in the paragraphs below. They all have polynomial complexity. Therefore, the overall runtime of the solver is polynomial too.

In order to compute messages between the inflow and the outflow subproblem, we apply Algorithm All-Lifted-MM-Out. Min-marginals for messages between the path subproblems and the in/outflow subproblems are obtained for one shared variable at the time. The same holds for exchanging messages between the cut subproblems and the in/outflow subproblems. This is done by calling restricted versions of optimization algorithms of the path and cut subproblems (Algorithms Path-Subproblem-Optimization and Cut-Subproblem-Optimization). For in/outflow subproblems, we use one call of Algorithm Opt-Out-Cost followed by either Algorithm Skip-One or Algorithm Backward-DFS limited to single variable reparametrization.

Messages between inflow and outflow subproblems. Algorithm All-Lifted-MM-Out contains many subroutines that employ a full or a partial DFS on all nodes reachable from the central node within the relevant time gap. In these cases, we use a precomputed node order instead of a complete DFS as described in the last paragraph of Section 5.4.1. One call of the full DFS (Algorithms Opt-Out-Cost and Backward-DFS) requires to process all vertices reachable from v within the maximal time gap for edge length. This comprises $L_{\max}$ video frames (we use $L_{\max} = 50$ or 60). Let us denote by D the maximum number of detections in one frame. The complete DFS processes maximally $DL_{\max}$ vertices. Incomplete DFS used in Algorithm Skip-One processes in each step vertices in L layers. In the worst case, this is done for all relevant frames in the distances $1, \dots, L_{\max}$. Processing one vertex requires checking its neighbors in the base graph. Their number is bounded by $KL_{\max}$ where $K = 3$. See the paragraph about sparsification in Section 5.6. Putting all together, the complexity of Algorithm All-Lifted-MM-Out for one subproblem is $\mathcal{O}(DL^3_{\max})$. We have two subproblems for each (lifted) graph vertex, yielding complexity $\mathcal{O}(|V'|DL^3_{\max})$ for sending messages between all inflow and outflow subproblems in one message passing iteration.

Messages from path subproblems. Obtaining min marginal for one edge variable of a path subproblem requires two calls of Algorithm Path-Subproblem-Optimization where the variable is restricted to be zero resp. one. Its complexity is linear in the number of path edges. So, min-marginals for all path edges are obtained in $\mathcal{O}(|P|^2)$.

Messages from cut subproblems. Min marginal of one variable of a cut subproblems is obtained by adjusting its optimization Algorithm Cut-Subproblem-Optimization. The complexity is given by the complexity of the employed linear assignment problem which can be solved in polynomial time.

Cutting plane procedures. The cutting plane algorithms are called each 20 iterations. We allow to add maximally $0.5 \cdot |\mathcal{S}_0|$ new factors during one separation call, where $\mathcal{S}_0$ is the initial set of subproblems containing only inflow and outflow factors. So it holds, $|\mathcal{S}_0| = 2|V'|$. Once added, the subproblems influence the runtime via taking part in the message passing (see Section 5.4.5).

Before discussing the complexity ot the cutting plane procedures, we provide bounds on the sizes of the used edge sets. There is maximally $KL_{\max}$ base edges and $(D(L_{\max}-1))$ lifted edges going from (resp. to) every node. The edge costs for separation are obtained as min marginals of inflow and outflow subproblems via Algorithm Separation-Costs. Therefore, there is maximally one negative base edge and maximally $L_{\max}$ negative lifted edges going from (resp. to) each node.

Cutting plane itself (Sections 5.4.2 and 5.4.3) contains sorting of subsets of base or lifted edges which has complexity $\mathcal{O}(|E^-|\log|E^-|)$ (resp. $\mathcal{O}(|E^+|\log|E^+|)$).

Cut subproblems separation. Using the arguments from the previous paragraph, we have the following complexity bounds for the edge sets in Algorithm Separate-Cut-Subproblem: $|E'^-| \in \mathcal{O}(|V|L_{\max})$ and $|E^+| \in \mathcal{O}(|V|L_{\max})$.

The search for candidate cut subproblems iterates over edges $ij \in E^+$ and inspects all combinations of predecessors of i and descendants of j. Each node stores maximally $DL_{\max}$ descendants resp. predecessors (see Line 4 in Connect). Therefore, the search has complexity $\mathcal{O}(|V|D^2L^3_{\max})$.

There is maximally one cut subproblem separated for each negative lifted edge in E'^-. Our procedure for creating cut (Line 13 in Algorithm Separate-Cut-Subproblem) searches all base edges that start in maximally $DL_{\max}$ vertices yielding complexity $\mathcal{O}(DL^2_{\max})$. Therefore, separating all cut subproblems has complexity $\mathcal{O}(|V|DL^3_{\max})$. This complexity is dominated by the above described search for cut candidates $\mathcal{O}(|V|D^2L^3_{\max})$.

Path subproblem separation. Using the arguments from the previous paragraphs, we have the following bounds for the edge sets in Algorithm Separate-Path-Subproblem: $|E^-| \in \mathcal{O}(|V|L_{\max})$ and $|E'^+| \in \mathcal{O}(|V|DL_{\max})$. We call Algorithms Inner-Paths and Outer-Paths for edges $ij \in E'^-$. In both algorithms, we inspect all pairs (p, d) of predecessors of i and descendants of j (resp. the other way round). The number of predecessors resp. descendants is bounded by $DL_{\max}$. Therefore, this search itself has complexity $\mathcal{O}(|V|D^2L^3_{\max})$.

Our path extraction procedure on Lines 4 and 5 in Inner-Paths has complexity $\mathcal{O}(DL^2_{\max})$. Line 3 ensures that the path extraction is called only if vertices p and d were not connected before. Therefore, it is called maximally once for each edge in E'^+ where $|E'^+| \in \mathcal{O}(|V|DL_{\max})$. This yields the complexity of all calls of Inner-Paths together $\mathcal{O}(|V|D^2L^3_{\max})$.

As opposed to Algorithm Inner-Paths, Outer-Paths does not limit the usage of each positive lifted edge $dp \in E'^+$ for the path extraction on Lines 4 and 5 to be maximally one. So, this leads to the worst case complexity of all calls of Outer-Paths $\mathcal{O}(|V|D^3L^5_{\max})$.

Note that we could achieve $\mathcal{O}(|V|D^2L^3_{\max})$ for all calls of Outer-Paths by bounding the usage of each lifted edge $dp \in E'^+$ by a constant. In fact, we allow only a bounded number of subproblems sharing one variable to be added from the subproblems queues to subproblems set $\mathcal{S}$. Moreover, if we have multiple subproblems containing certain edge $dp \in E'^+$, those separated earlier lead to a better lower bound improvement. The reason is that we traverse negative edges from E^- in ascending order on Line 7 in Separate-Path-Subproblem. However, the calls of Outer-Paths did not turn out to be a bottleneck in practice, so we do not use this limitation in our implementation.

Primal solution. We compute a new primal solution in every five iterations. We use Algorithm Init-MCF for obtaining base edge costs. Then, we use successive shortest paths algorithm for solving minimum cost flow problem and finally local search heuristic given by Algorithm Local-Search, see Section 5.3.6. The complexity of solving (MCF) is the complexity of successive shortest path algorithm which is polynomial. Local search heuristic requires to compute and update cumulative costs between candidate paths. They can be computed in time linear in the number of lifted edges $\mathcal{O}(|E'|)$. (MCF) costs are obtained by calling Algorithm Opt-Out-Cost. Its complexity is discussed above.

5.6 EXPERIMENTS

We integrate our approximate LDP solver into an MOT system (Figure 5.8) and show on challenging datasets that higher order MOT is scalable to big problem instances. In the next sections, we describe our experimental setup and present results. We clarify the edge cost calculation and the construction of the base and the lifted graph and their sparsification.

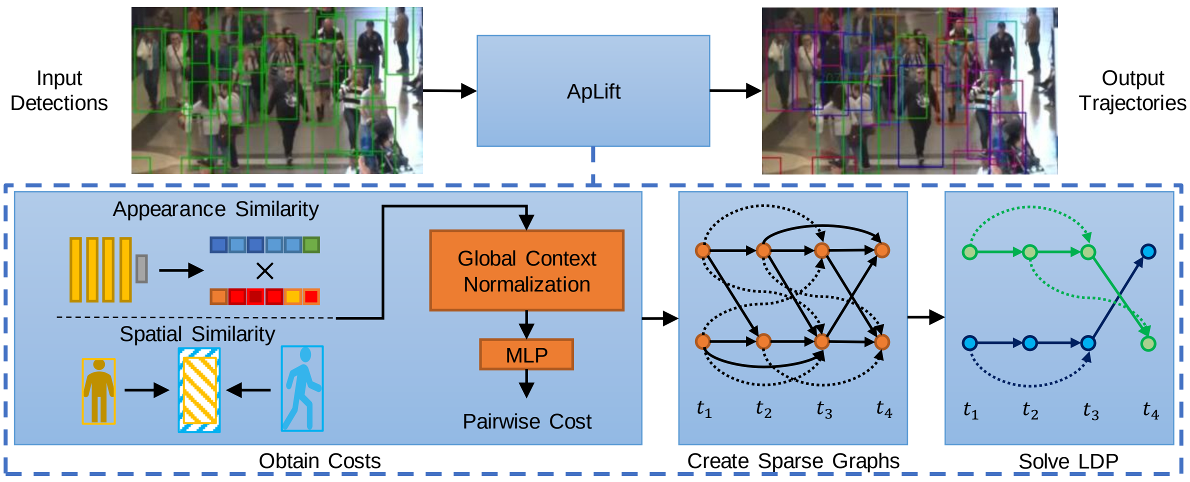


Figure 5.8: Overview of the ApLift framework. Input detections are used to obtain pairwise costs by an MLP with spatial and appearance features. Based on the costs, two sparse graphs are constructed and passed to our proposed approximate LDP solver. Dashed arrows represent lifted edges and solid arrows base edges. In figure *Solve LDP* equally colored nodes and edges belong to the same trajectory.

5.6.1 PAIRWISE COSTS

We use multi-layer perceptron (MLP) to predict the likelihood that two detections belong to the same trajectory. We divide the maximal frame distance into 20 intervals of equal length and train one separate MLP for each set of frame distances. We transform the MLP output to the cost of the edge between the detections and use it in our (LDP) objective. A negative cost indicates that two detections belong to the same trajectory. A positive cost reflects the opposite.

MLP architecture. Each MLP consists of a fully connected layer with the same number of neurons as the input size, followed by a LeakyReLU activation (Maas *et al.*, 2013) and a fully connected single neuron output layer. We add sigmoid activation in the training. We describe our spatial and visual features used as the input in the paragraphs below.

The spatial feature uses bounding box information of two detections v and w. We align the boxes such that their centers overlap. The similarity feature $\mathfrak{f}_{\text{spa}}(vw) \in [0, 1]$ is the intersection-over-union between two aligned boxes.

Appearance feature. We create an appearance feature F_v for each detection v by training the method from Zheng *et al.* (2019) on the training set of the respective benchmark and additional data from Zheng *et al.* (2015); Wei *et al.* (2018); Ristani *et al.* (2016b). The similarity feature $\mathfrak{f}_{\text{app}}(vw)$ between detection v and w given by $\mathfrak{f}_{\text{app}}(vw) := \max\{0, \langle F_v, F_w \rangle\}$ is used. A higher value indicates higher similarity.

Global context normalization. The two features $\mathfrak{f}_{\text{spa}}(vw)$, $\mathfrak{f}_{\text{app}}(vw)$ depend entirely on the nodes v and w. To include global context, we append several normalized versions of the two features to the edge feature vector, inspired by our approach from Chapter 4. Both features $\mathfrak{f}_{\text{spa}}(vw)$, $\mathfrak{f}_{\text{app}}(vw)$ of edge vw undergo a five-way normalization. In each case, the maximum feature value from a relevant set of edges is selected as the normalization value. The normalization is done by dividing the two features $\mathfrak{f}_{\text{spa}}(vw)$, $\mathfrak{f}_{\text{app}}(vw)$ by each of their five normalization values. This yields 10 values. Another set of 10 values for edge vw is obtained by dividing $\mathfrak{f}^2_{\text{spa}}(vw)$, $\mathfrak{f}^2_{\text{app}}(vw)$ by each of the five normalization values. Together with the two not normalized feature values, the edge feature vector has a length 22. See Appendix of Hornakova *et al.* (2021) for details.

Training. We iteratively train our MLP on batches B containing sampled edges. To compensate for the imbalance between true positive and true negative edges, we use an α-balanced focal loss (Lin *et al.*, 2018) with $\gamma = 1$. We define the α-weight $\alpha^{(g,L)}$ to weight the correct classification of edge vw with the ground truth flow value $g_{vw} \in \{0, 1\}$, time distance L between v in frame f_v and w in frame f_w, and value $g \in \{0, 1\}$ via $\alpha^{(g,L)} := 1/|\{vw \in E|\ g_{vw} = g, |f_v - f_w| = L\}|$. We optimize the classifier using Adam with $l_r = 0.1$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. To reduce complexity while maintaining variety during training, we introduce an extended sampling. Given a frame f, we create batches $B(f)$ by sampling detections from a fixed sequence of frame shifts starting at frame f ensuring that all temporal distances L are present in $B(f)$ (details in the Appendix of Hornakova *et al.* (2021)). We then subsample the k-nearest detections to a randomly generated image position with $k = 160$, which sensitizes training to crowded scenes. We train the MLP for 3 epochs with batches $B(f)$ for all frames f of the dataset.

5.6.2 GRAPH CONSTRUCTION

We create the base and the lifted graph edges between detections with time distances up to 2 seconds. We also add an edge from source s, and to sink t to each detection. In order to reduce computational complexity, we apply sparsification on both base and lifted graph as described later.

Costs. We obtain base and lifted costs c and c' from the same MLP classifier (Section 5.6.1). Due to decreasing classification accuracy with increasing frame distance L, we multiply the costs by a decay weight $\omega_L = (10 \cdot L + 0.1)^{-1}$, so that edges representing long temporal distances have lower weights. Edges from s and to t have costs zero.

Finally, we use simple heuristics to find pairs that are obviously matching or non-matching. We set the corresponding costs to be high in absolute value, negative for matching and positive for non-matching, thereby inducing soft constraints. An obvious match is given by a nearly maximal feature similarity. Detection pairs are obviously non-matching if the displacement between their bounding boxes is too high. See the Appendix of Hornakova *et al.* (2021) for details.

	Method	MOTA↑	IDF1↑	MT↑	ML↓	FP↓	FN↓	IDS↓	Frag↓	Frames	Density
MOT20	ApLift (ours)	**58.9**	56.5	**513**	**264**	17739	**192736**	2241	2112	1119.8	170.9
	MPNTrack	57.6	**59.1**	474	279	16953	201384	**1210**	**1420**		
	Tracktor++v2	52.6	52.7	365	331	**6930**	236680	1648	4374		
MOT17	CTTrackPub	**61.5**	59.6	621	752	**14076**	200672	2583	4965	845.6	31.8
	ApLift (ours)	60.5	**65.6**	**798**	**728**	30609	**190670**	1709	2672		
	LifT (ours)	60.5	**65.6**	637	791	14966	206619	1189	3476		
	MPNTrack	58.8	61.7	679	788	17416	213594	**1185**	**2265**		
MOT16	ApLift (ours)	**61.7**	**66.1**	**260**	**237**	9168	**60180**	495	802	845.6	30.8
	LifT (ours)	61.3	64.7	205	258	4844	65401	389	1034		
	MPNTrack	58.6	61.7	207	258	4949	70252	**354**	**684**		
	GSM	57.0	55.0	167	262	**4332**	73573	475	859		
MOT15	LifT (ours)	**52.5**	**60.0**	244	186	6837	21610	730	1047	525.7	10.8
	MPNTrack	51.5	58.6	225	187	7260	21780	**375**	**872**		
	ApLift (ours)	51.1	59.0	**284**	**163**	10070	**19288**	677	1022		
	Tracktor15	44.1	46.7	130	189	**6477**	26577	1318	1790		

Table 5.1: Comparison of ApLift with the best performing trackers w.r.t. MOTA metric on the MOT challenge. ↑ higher is better, ↓ lower is better. The two rightmost columns: average number of frames per sequence and the average number of detections per frame for the dataset. References: MPNTrack (Braso and Leal-Taixe, 2020), Tracktor++v2 (Bergmann *et al.*, 2019), CTTrackPub (Zhou *et al.*, 2020), GSM (Liu *et al.*, 2020), Tracktor15 (Bergmann *et al.*, 2019).

Sparsification. The base edges are an intersection of two edge sets. The first set contains for every $v \in V'$ edges to its 3 nearest (lowest-cost) neighbors from every subsequent time frame. The second set selects for every vertex the best edges to its preceding frames analogically. Moreover, edges longer than 6 frames must have costs lower than 3.0. To avoid double counting of edge costs, we subsequently set costs of all base edges between non-consecutive frames to zero, so that only lifted edges maintain the costs. If a lifted edge has cost around zero, it is not discriminative and we remove it, unless it overlaps with a (zero-valued) base edge.

5.6.3 INFERENCE

For a fair comparison to state-of-the-art, we filter and refine detections using Tracktor (Bergmann *et al.*, 2019) as in Chapter 4. Different to Chapter 4, we apply Tracktor to recover missing detections before running the LDP solver.

While we solve MOT15/16/17 on global graphs representing the complete sequences, we solve MOT20 in time intervals in order to decrease memory consumption and runtime.

Interval Solution. First, we solve the problem on independent non-overlapping adjacent intervals and fix the trajectories in the interval centers. Second, we solve the problem on a new set of intervals where each of them covers unassigned detections in two initial neighboring intervals and enables connections to the fixed trajectory fragments. The cost of a connection between a detection and a trajectory fragment is obtained as the sum of costs between the unassigned detection and the detections within the trajectory fragment.

The initial intervals have the following form: $[il+1, (i+1)l]$ for $i \in \{0, 1 \ldots, n\}$, where $l = 3 \cdot L_{\max}$. We fix trajectories in the centers of the initial intervals $[il + L_{\max} + 1, (i+1)l - L_{\max}]$ for $i \in \{1, \ldots, n-1\}$.

We use the maximal edge length $L_{\max} = 50$ frames in MOT20. Therefore, $l = 150$ is the minimal interval length such that all edges from a detection are used when assigning the detection to a trajectory. This way, the solver has sufficient context for making each decision. Intervals longer than 200 frames increase the complexity significantly for MOT20, therefore we use interval length 150 in our experiments.

Post-processing. We use simple heuristics to check if base edges over long time gaps correspond to plausible motions, and split trajectories if necessary. Finally, we use linear interpolation to recover missing detections within a trajectory. Appendix of Hornakova *et al.* (2021) contains further details on inference.

5.6.4 TRACKING EVALUATION

We evaluate our method ApLift on four standard MOT benchmarks. The MOT15/16/17 benchmarks (Leal-Taixé *et al.*, 2015; Milan *et al.*, 2016) contain semi-crowded video sequences filmed from a static or moving camera. MOT20 (Dendorfer *et al.*, 2020) comprises crowded scenes with a considerably higher number of frames and detections per frame, see Table 5.1. The challenge does not come only with the data size. Detectors make more errors in crowded scenes due to frequent occlusions and appearance features are less discriminative as the distance of people to the camera is high. Using higher order information helps in this context. However, the number of lifted edges grows quadratically with the number of detections per frame. Therefore, it is crucial to make the tracker scalable to these massive instances. We use the following ingredients to solve the problems: (i) fast but accurate method for obtaining edge costs, (ii) approximate LDP solver delivering high-quality results fast, (iii) preprocessing heuristics, (iv) interval solution keeping sufficient context for each decision.

We use the training data of the corresponding dataset for training and the public detections for training and test.

We compare our method ApLift using standard MOT metrics. MOTA (Bernardin and Stiefelhagen, 2008b) and IDF1 (Ristani *et al.*, 2016a) are considered the most representative as they incorporate other metrics (in particular recall and precision). IDF1 is more penalized by inconsistent trajectories. We also report mostly tracked (MT) and mostly lost trajectories (ML), false negatives (FN) and false positives (FP), ID switches (IDS) and fragmentation (Frag) as provided by the evaluation protocols (Bernardin and Stiefelhagen, 2008b) of the benchmarks.

Table 5.1 shows the comparison to the best (w.r.t. MOTA) peer-reviewed methods on test sets. Our approximate solver achieves almost the same results on MOT15/16/17

E'	MP steps	Base cost	IDF1↑	MOTA↑	FP↓	FN↓	IDS↓
Dense	82	Zero	**71.0**	**66.3**	2826	**109263**	1369
Dense	0	Zero	70.3	**66.3**	2832	109265	1354
Dense	82	Orig.	69.8	**66.3**	**2824**	109266	1355
Sparse	82	Orig.	69.1	**66.3**	2825	**109263**	**1316**

Table 5.2: Influence of lifted graph sparsification, message passing (MP) steps and using zero base costs on MOT17 train without postprocessing.

as tracker LifT from Chapter 4 which uses the optimal LDP solver and more complex features.

Overall, ApLift performs on par with state-of-the-art (at the time of the method publication) on all evaluated benchmarks, especially in MOTA and IDF1. Our complete results and videos are publicly available at the MOT challenge website https://motchallenge.net/method/MOT=4031&chl=13. Detailed results of our tracker ApLift on individual sequences are in Table 5.5. The proposed method achieves overall low FN values but slightly high FP values. FP/FN are mostly affected by preprocessing the input detections and interpolation in the post-processing.

Table 5.2 shows the influence of various settings on the performance of MOT17 train. While we usually set the base edge costs to zero (Section 5.6.2), we need to keep them when using the sparsified lifted graph. Both, message passing and dense lifted edges improve IDF1 and IDS. However, MOTA, FN, and FP remain almost unchanged.

5.6.5 COMPARISON WITH LIFT

In this section, we compare the runtime of ApLift against LifT. ApLift can compute one interval of MOT20 (150 frames) or an entire sequence of MOT17 with less than 20GB RAM, using a single CPU core.

First, we compare solver ApLift with the two-step version of LifT for subsequences of MOT20 in Tables 5.3 and 5.4. With increasing problem complexity, ApLift outperforms LifT w.r.t. runtime while achieving similar IDF1. Counter-intuitively, as we progress towards increasingly better optimization objective values, the tracking metrics can slightly decrease due to imperfect edge costs. We compare ApLift against optimal (one step) LifT on MOT17 train in Table 5.6. We discuss the details of the experimental settings in the following paragraphs.

LifT global solution vs. two-step procedure. LifT is based on ILP solver Gurobi. It solves (LDP) optimally. However, it is often not able to solve the problem on the full graphs. Therefore, LifT uses a two-step procedure (Section 4.7.1). First, solutions are found on small time intervals to create tracklets. Second, the problem is solved on tracklets. This approach simplifies the problem significantly but the delivered solutions are not globally optimal anymore. We have observed that using our new input costs,

n	Measure	LifT	ApLift 6	ApLift 11	ApLift 31	ApLift 51
50	IDF1↑	80.6	**83.3**	**83.3**	81.5	81.5
	time [s]	272	2	4	16	35
100	IDF1↑	80.4	**82.5**	**82.5**	81.6	81.6
	time [s]	484	14	24	97	218
150	IDF1↑	78.1	**81.0**	**81.0**	79.8	79.8
	time [s]	1058	25	46	192	431
200	IDF1↑	73.2	**75.4**	**75.4**	74.6	74.6
	time [s]	2807	36	66	277	616

Table 5.3: Runtime and IDF1 comparison of our LDP solvers: ApLift (Chapter 5) with 6, 11, 31, and 51 iterations and the two-step procedure of LifT (Chapter 4) on the first n frames of sequence *MOT20-01* from MOT20.

LifT is able to solve some problem sequences globally without the two-step procedure. Therefore, we compare ApLift with LifT using both the two-step procedure and the global solution.

Influence of input costs. Our input data contain many soft constraints for obviously matching pairs of detections. Those are edges with negative costs significantly higher in absolute value than other edges costs. LifT finds an initial feasible solution using only base edges. This solution may be already very good due to the costs of obviously matching pairs. Moreover, Gurobi contains a lot of efficient precomputing steps, so it can recognize that the respective variables should be active in the optimum and reduce the search space.

Parameters. We adjust the parameters of ApLift to work with comparable data as LifT. For instance, we do not set the cost of base edges to zero (as described in Section 5.6.2) because LifT does not enable this option. So, the costs of overlapping base and lifted edges are duplicated as opposed to most of the other experiments. Note that this setting is less convenient for ApLift (see Table 5.2). Moreover, if there is no edge between two detections within the maximal time distance in the input data, we can add a lifted edge with a high positive cost for such pair in ApLift. This is useful for reducing the input size for MOT20 dataset. LifT does not have this option. Therefore, we disable this option for ApLift too.

Subsequences of MOT20. We present a comparison between ApLift and LifT using two-step procedure on subsequences of MOT20-01 in Table 5.3. Here, ApLift is faster and has even slightly better IDF1 score than LifT. In Table 5.4, we present a comparison on first n frames of sequence MOT20-02 where LifT finds solutions faster than ApLift using many iterations. We assume that this is caused by the input costs

n	Measure	LifT	ApLift 6	ApLift 11	ApLift 31	ApLift 51
50	IDF1↑	**83.4**	**83.4**	**83.4**	**83.4**	**83.4**
	time [s]	62	4	7	25	46
100	IDF1↑	**80.6**	79.9	79.9	79.9	79.9
	time [s]	124	30	54	182	360
150	IDF1↑	**78.7**	76.8	76.8	76.8	76.8
	time [s]	222	61	110	378	780
200	IDF1↑	**77.6**	75.8	75.8	75.8	75.8
	time [s]	354	95	177	604	1195

Table 5.4: Runtime and IDF1 comparison of our LDP solvers: ApLift (Chapter 5) with 6, 11, 31, and 51 iterations and the two-step procedure of LifT (Chapter 4) on first n frames of sequence *MOT20-02* from MOT20.

that are convenient for Gurobi, see the discussion above.

Train set of MOT17. We compare ApLift with LifT on global training sequences of MOT17. That is, we do not use the two-step procedure. Therefore, LifT finds the globally optimal solution if it finishes successfully. The runtime of LifT is exponential in general and it can be often killed because of memory consumption if run on global sequences. Therefore, we perform these experiments on a machine having 2000 GB RAM and multiple CPUs each having 64 cores.

The results are in Table 5.6. Asterisks in LifT time column indicate that the problem cannot be finished. Some of the processes were killed by the system because of too much memory consumption. Some processes did not finish within more than 27 hours. Moreover, LifT often occupied up to 30 cores for solving one sequence. ApLift uses only one core. In cases when LifT does not finish, we evaluate the best feasible solution found by LifT. Those were typically the initial feasible solutions. That is, the solutions that ignore the lifted edges. Obtaining the initial solutions for these difficult instances took between 1700 and 4600 seconds. The numbers in brackets relate ApLift results to LifT results. The time column provides the ratio between ApLift time and LifT time. The IDF1 column presents the difference between ApLift and LifT.

	Sequence	MOTA↑	IDF1↑	MT↑	ML↓	FP↓	FN↓	IDS↓	Frag. ↓
MOT20 Train	MOT20-01	65.8	62.0	31	10	180	6512	109	87
	MOT20-02	62.3	55.1	108	18	1393	56420	548	534
	MOT20-03	80.4	76.1	427	66	5427	55552	623	591
	MOT20-05	74.6	57.8	643	115	8778	152927	2231	2063
	OVERALL	74.4	62.8	1209	209	15778	271411	3511	3275
MOT20 Test	MOT20-04	79.3	68.8	412	40	8315	47364	968	840
	MOT20-06	36.1	36.8	41	111	4786	79313	740	744
	MOT20-07	56.9	54.7	40	15	936	13135	194	195
	MOT20-08	26.5	33.8	20	98	3702	52924	339	333
	OVERALL	58.9	56.5	513	264	17739	192736	2241	2112
MOT17 Train	MOT17-02-DPM	42.2	52.5	14	29	125	10588	26	26
	MOT17-02-FRCNN	47.3	58.4	15	21	227	9532	27	30
	MOT17-02-SDP	55.1	60.5	17	16	289	7994	53	52
	MOT17-04-DPM	70.9	78.9	40	21	340	13481	17	29
	MOT17-04-FRCNN	68.0	78.4	39	21	179	15044	5	13
	MOT17-04-SDP	77.9	80.8	47	13	439	10035	29	68
	MOT17-05-DPM	60.0	64.5	48	34	475	2260	31	24
	MOT17-05-FRCNN	57.8	64.0	55	32	650	2225	46	41
	MOT17-05-SDP	62.6	67.8	59	19	693	1842	53	46
	MOT17-09-DPM	73.0	72.8	14	1	46	1380	10	9
	MOT17-09-FRCNN	71.5	68.4	14	1	105	1403	10	9
	MOT17-09-SDP	74.1	72.9	14	1	66	1302	10	11
	MOT17-10-DPM	65.3	67.4	32	6	847	3545	61	74
	MOT17-10-FRCNN	62.8	65.8	40	2	2121	2513	139	114
	MOT17-10-SDP	66.3	66.5	43	2	1967	2189	173	120
	MOT17-11-DPM	69.2	75.7	34	21	248	2624	37	17
	MOT17-11-FRCNN	71.5	76.8	38	18	412	2233	47	15
	MOT17-11-SDP	72.6	78.5	42	13	547	1981	58	19
	MOT17-13-DPM	64.4	64.8	55	33	627	3436	83	56
	MOT17-13-FRCNN	67.8	63.4	77	8	1739	1892	120	76
	MOT17-13-SDP	67.2	63.7	72	18	1388	2312	117	60
	OVERALL	66.0	71.4	809	330	13530	99811	1152	909
MOT17 Test	MOT17-01-DPM	48.8	54.3	8	10	113	3181	8	21
	MOT17-01-FRCNN	47.0	57.5	9	10	360	3050	11	21
	MOT17-01-SDP	45.2	55.4	9	10	488	3033	13	29
	MOT17-03-DPM	73.8	73.4	85	17	4360	22905	118	261
	MOT17-03-FRCNN	72.8	74.7	74	17	3471	24883	109	234
	MOT17-03-SDP	77.7	75.4	94	13	4676	18482	139	386
	MOT17-06-DPM	57.7	61.2	94	76	1142	3765	77	91
	MOT17-06-FRCNN	57.3	58.4	102	59	1652	3279	102	140
	MOT17-06-SDP	57.2	59.5	107	58	1700	3251	87	125
	MOT17-07-DPM	45.7	52.5	11	15	1062	8038	80	126
	MOT17-07-FRCNN	45.0	53.1	11	15	1345	7862	75	135
	MOT17-07-SDP	46.6	53.8	13	11	1622	7310	87	166
	MOT17-08-DPM	33.7	44.3	17	37	421	13533	48	67
	MOT17-08-FRCNN	31.5	42.1	17	37	462	13948	53	74
	MOT17-08-SDP	34.5	45.2	18	34	445	13339	63	85
	MOT17-12-DPM	47.6	61.9	23	36	563	3959	20	32
	MOT17-12-FRCNN	47.8	62.3	18	40	296	4219	13	24
	MOT17-12-SDP	50.0	66.1	19	42	488	3836	11	31
	MOT17-14-DPM	37.8	51.0	19	71	1147	10191	151	150
	MOT17-14-FRCNN	33.9	48.4	25	62	2369	9636	206	228
	MOT17-14-SDP	37.0	49.9	25	58	2427	8970	238	246
	OVERALL	60.5	65.6	798	728	30609	190670	1709	2672

Table 5.5: Evaluation results for training and test sequences for datasets MOT17 (Milan *et al.*, 2016) and MOT20 (Dendorfer *et al.*, 2020)

	LifT		ApLift 6		ApLift 11		ApLift 31		ApLift 51		ApLift 101	
Sequence	Time↓	IDF1↑	Time↓	IDF1↑	Time↓	IDF1↑	Time↓	IDF1↑	Time↓	IDF1↑	Time↓	IDF1↑
02-DPM	7324	**49.4**	94	47.4	157	47.4	513	49.1	989	49.1	2415	49.1
	(1.0)	(0.0)	(0.01)	(−2.00)	(0.02)	(−2.00)	(0.07)	(−0.30)	(0.14)	(−0.30)	(0.33)	(−0.30)
02-FRCNN	4073	54.7	97	**54.9**	161	**54.9**	526	**54.9**	1021	**54.9**	2503	**54.9**
	(1.0)	(0.0)	(0.02)	(0.20)	(0.04)	(0.20)	(0.13)	(0.20)	(0.25)	(0.20)	(0.61)	(0.20)
02-SDP	7795	**56.7**	131	55.0	219	55.0	717	55.0	1410	55.0	3685	55.0
	(1.0)	(0.0)	(0.02)	(−1.70)	(0.03)	(−1.70)	(0.09)	(−1.70)	(0.18)	(−1.70)	(0.47)	(−1.70)
04-DPM	$*$	**75.4**	449	74.7	756	74.7	2220	74.7	3929	75.0	8578	75.0
	($*$)	(0.0)	($*$)	(−0.70)	($*$)	(−0.70)	($*$)	(−0.70)	($*$)	(−0.40)	($*$)	(−0.40)
04-FRCNN	4889	**79.2**	383	78.1	644	78.1	1811	76.3	3111	78.2	6565	78.2
	(1.0)	(0.0)	(0.08)	(−1.10)	(0.13)	(−1.10)	(0.37)	(−2.90)	(0.64)	(−1.00)	(1.34)	(−1.00)
04-SDP	$*$	**82.3**	499	78.0	839	78.0	2441	78.0	4294	77.7	9269	79.9
	($*$)	(0.0)	($*$)	(−4.30)	($*$)	(−4.30)	($*$)	(−4.30)	($*$)	(−4.60)	($*$)	(−2.40)
05-DPM	535	**65.0**	10	62.6	15	62.6	57	63.5	116	63.5	298	63.5
	(1.0)	(0.0)	(0.02)	(−2.40)	(0.03)	(−2.40)	(0.11)	(−1.50)	(0.22)	(−1.50)	(0.56)	(−1.50)
05-FRCNN	514	**66.6**	10	63.8	15	63.8	57	64.0	118	63.9	315	65.6
	(1.0)	(0.0)	(0.02)	(−2.80)	(0.03)	(−2.80)	(0.11)	(−2.60)	(0.23)	(−2.70)	(0.61)	(−1.00)
05-SDP	604	**67.9**	11	**67.9**	18	**67.9**	67	67.1	137	67.1	364	67.6
	(1.0)	(0.0)	(0.02)	(0.00)	(0.03)	(0.00)	(0.11)	(−0.80)	(0.23)	(−0.80)	(0.60)	(−0.30)
09-DPM	6692	**67.5**	42	66.4	70	66.4	232	**67.5**	480	**67.5**	1281	**67.5**
	(1.0)	(0.0)	(0.01)	(−1.10)	(0.01)	(−1.10)	(0.03)	(0.00)	(0.07)	(0.00)	(0.19)	(0.00)
09-FRCNN	11888	**68.2**	37	**68.2**	61	**68.2**	201	**68.2**	407	**68.2**	1095	**68.2**
	(1.0)	(0.0)	(0.00)	(0.00)	(0.01)	(0.00)	(0.02)	(0.00)	(0.03)	(0.00)	(0.09)	(0.00)
09-SDP	1462	**68.6**	44	67.1	74	67.1	247	68.5	512	68.5	1443	68.5
	(1.0)	(0.0)	(0.03)	(−1.50)	(0.05)	(−1.50)	(0.17)	(−0.10)	(0.35)	(−0.10)	(0.99)	(−0.10)
10-DPM	$*$	66.0	279	**68.0**	466	**68.0**	1524	66.8	3087	67.0	9478	67.9
	($*$)	(0.0)	($*$)	(2.00)	($*$)	(2.00)	($*$)	(0.80)	($*$)	(1.00)	($*$)	(1.90)
10-FRCNN	$*$	65.2	310	68.8	511	68.5	1689	**69.4**	3428	**69.4**	10743	**69.4**
	($*$)	(0.0)	($*$)	(3.60)	($*$)	(3.30)	($*$)	(4.20)	($*$)	(4.20)	($*$)	(4.20)
10-SDP	$*$	65.4	379	67.0	630	67.0	2090	67.4	4294	67.1	13379	**69.8**
	($*$)	(0.0)	($*$)	(1.60)	($*$)	(1.60)	($*$)	(2.00)	($*$)	(1.70)	($*$)	(4.40)
11-DPM	1991	**76.3**	60	**76.3**	99	**76.3**	335	**76.3**	672	**76.3**	1672	**76.3**
	(1.0)	(0.0)	(0.03)	(0.00)	(0.05)	(0.00)	(0.17)	(0.00)	(0.34)	(0.00)	(0.84)	(0.00)
11-FRCNN	2382	**78.3**	68	**78.3**	113	**78.3**	366	**78.3**	729	**78.3**	1799	**78.3**
	(1.0)	(0.0)	(0.03)	(0.00)	(0.05)	(0.00)	(0.15)	(0.00)	(0.31)	(0.00)	(0.76)	(0.00)
11-SDP	3195	80.0	68	79.8	113	79.8	370	**80.1**	748	80.0	2057	80.0
	(1.0)	(0.0)	(0.02)	(−0.20)	(0.04)	(−0.20)	(0.12)	(0.10)	(0.23)	(0.00)	(0.64)	(0.00)
13-DPM	$*$	62.8	152	**66.8**	252	**66.8**	944	**66.8**	2008	**66.8**	6340	65.7
	($*$)	(0.0)	($*$)	(4.00)	($*$)	(4.00)	($*$)	(4.00)	($*$)	(4.00)	($*$)	(2.90)
13-FRCNN	$*$	62.5	217	**69.8**	351	**69.8**	1331	**69.8**	2942	67.7	9813	66.2
	($*$)	(0.0)	($*$)	(7.30)	($*$)	(7.30)	($*$)	(7.30)	($*$)	(5.20)	($*$)	(3.70)
13-SDP	$*$	64.5	196	**66.8**	326	**66.8**	1237	**66.8**	2698	66.2	8954	65.6
	($*$)	(0.0)	($*$)	(2.30)	($*$)	(2.30)	($*$)	(2.30)	($*$)	(1.70)	($*$)	(1.10)
OVERALL	$*$	**70.7**	168	70.3	280	70.3	904	70.2	1768	70.3	4859	**70.7**
	($*$)	(0.0)	($*$)	(−0.40)	($*$)	(−0.40)	($*$)	(−0.50)	($*$)	(−0.40)	($*$)	(0.00)

Table 5.6: Runtime and IDF1 comparison of our LDP solvers: ApLift with 6, 11, 31, 51 and 101 iterations and globally optimal (one step) LifT on MOT17 train. Numbers in parenthesis in the time column show the difference between the solvers, in the IDF1 column the ratio between Lift and ApLift.

6 CONCLUSIONS

This thesis studies the enhancement of two graph partitioning optimization problems via lifted edges that represent pairwise connectivity priors between graph nodes. The first is an extension of the multicut problem (MC) called lifted multicut (LMC) which represents a decomposition of an undirected graph into connected components. The second model is the lifted disjoint paths (LDP) problem, an extension of the disjoint paths problem (DP), which represents a decomposition of a directed graph into a set of node-disjoint paths. While (LMC) motivated especially by solving the task of image segmentation has been known before, the concept of (LDP) is our contribution.

The advantage of the enhancements is the possibility to include more information into the problems without changing their sets of feasible solutions. Moreover, these long-range connectivity priors are often more reliable than the short-range information encoded by the original models. This thesis shows that the extension of (DP) via lifted edges is beneficial because it leads to better results in the studied applications in comparison to the methods applying the original model. In particular, our first proposed method LifT for multiple object tracking (MOT) based on our optimal LDP solver significantly outperformed state-of-the-art trackers on three standard MOT datasets at the time of its publication. Our second method ApLift based on our approximate LDP solver achieved comparable or better results to the state-of-the-art methods including LifT at four standard MOT benchmarks at the time of its publication. Moreover, ApLift is applicable to larger sequences than LifT.

This is in compliance with the observations made within numerous applications of lifted multicut problem to computer vision tasks where this model using lifted edges typically leads to better results in comparison to methods using standard multicut.

We conclude that the big downside of these enhanced models is their increased complexity. We provide several proofs that certain tasks that are polynomially solvable for the original models become NP-complete or even NP-hard in the enhanced models. Importantly for the main part of this work, we show the extension of the polynomially solvable (DP) via lifted edges leads to an NP-hard problem (LDP). As opposed to (DP), the optimization problem (MC) is known to be NP-hard even without the extension via lifted edges. On the other hand, we show that while checking the consistency of a partial characterization of multicut can be done efficiently, the same task for lifted multicut is NP-complete. Similarly, we prove that deciding whether a partial characterization is maximally specific can be done in polynomial time for multicut while it is NP-hard to do so for the lifted multicut.

Despite the theoretical NP-hardness of the studied problems, they can be applied to some computer vision tasks. The reason is that the real data typically have a convenient structure that enables to obtain high-quality results w.r.t. the required task metrics even if only approximate methods are used for the solution. In case of (lifted) multicut, most of the methods rely on heuristics or approximate solvers. However, globally optimal solvers exist. We provide an overview of the solution methods and applications of (lifted) multicut in Chapter 2.

The main application of our proposed model (LDP), is multiple object tracking (MOT). We demonstrate that it is possible to apply (LDP) to MOT instances originating from real-world video sequences if we use high-quality edge costs. We demonstrate that our optimal LDP solver can be applied not only to graphs originating from small time intervals of MOT videos but sometimes also to global graphs representing the whole video sequences. This observation highlights the importance of three ingredients that are necessary for solving such problem instances optimally. First, we use state-of-the-art ILP solver Gurobi (Gurobi Optimization, 2019). Second, we compute high-quality edge costs. Third, we separate non-trivial ILP constraints that ensure a tighter LP relaxation than their simple variants and thus lead to a significant speed-up of the branch and bound procedure in Gurobi.

Whenever the size or complexity of the data prohibits the usage of our globally optimal solver, we can use one of the approximate methods that we propose. These are our approximate message passing solver and the two-step procedure of the optimal solver. They are able to produce results reasonably fast without compromising the solution quality w.r.t. the MOT metrics. We demonstrate that the NP-hard (LDP) model is applicable even for processing massive and crowded video sequences of the MOT20 dataset (Dendorfer *et al.*, 2020). Processing crowded videos is challenging for (LDP) considering that the number of lifted edges grows quadratically with the number of detections within one time frame. The combination of our approximate LDP solver, efficiently computable edge costs, and a subdivision of data keeping sufficient context for each decision make the solution possible.

SYMBOLS

$\{0,1\}^E$	In general, $x \in A^B$ denotes a mapping $x : B \to A$. Therefore, $x \in \{0,1\}^E$ assigns each edge $e \in E$ value 0 or 1.	7
$\mathbb{1}$	Vector of ones	32
$\mathbb{0}$	Vector of zeros	84
$-=$	Update the variable on the left-hand side by subtracting the formula on the right-hand side from it.	83
$+=$	Update the variable on the left-hand side by adding the formula on the right-hand side to it.	83
$\delta^-_E(v)$	Inflow edges of node v in $G = (V, E)$	84
$\delta^-_{E'}(v)$	Inflow edges of node v in $G' = (V', E')$	84
$\delta^+_E(v)$	Outflow edges of node v in $G = (V, E)$	84
$\delta^+_{E'}(v)$	Outflow edges of node v in $G' = (V', E')$	84
$\lambda_{GG'}$	Composed mapping $\phi_{G'} \circ \phi_G^{-1}$, lifting of multicuts from G to G'	27
ϕ_G	A bijection from D_G to M_G	25
$\Sigma_{GG'}(vw, C)$	The convex hull of $S_{GG'}(vw, C)$	42
θ^{s}	Real-valued vector defining objective of subproblem s	83
Ξ_G	The multicut polytope of G	39
$\Xi_{GG'}$	The lifted multicut polytope w.r.t. G and G'	39, 42
c	Cost vector of base edges in (LDP), cost vector of all edges in (LMC)	13, 39, 54
c'	Cost vector of lifted edges in (LDP)	54
$\mathrm{cl}_{GG'}\,\tilde{x}$	Closure of $\tilde{x}$ w.r.t. G and G' (Definition 3.7)	33
d	Cost vector of nodes in (LDP)	54
$d^1_{E'}$	Pseudo-metric on $X_{GG'}$	35
d^1_e	An auxiliary symbol for defining pseudo-metric on $X_{GG'}$	35
$E'[\tilde{x}]$	Edges of $G' = (V, E')$ decided by partial characterization $\tilde{x}$	31
D_G	The set of all decompositions of G	25
$\mathsf{d}(\mathsf{s})$	The length of the vectors in $\mathcal{X}^{\mathsf{s}}$	83
$\left(d^{\mu^n}_{E[n]}\right)_{n\in\{1,\ldots,\delta_G\}}$	The spectrum of pseudo-metrics on decompositions of G	36
$d^\mu_{E'}$	Pseudo-metric on $X_{GG'}$	35
$\mathrm{dom}\,\tilde{x}$	For any $\tilde{x} \in \{0,1,*\}^E$, $\mathrm{dom}\,\tilde{x} := \tilde{x}^{-1}(\{0,1\})$.	28

$d^\theta_{E'}$	A metric on $\hat{X}_{GG'}$	37
d^θ_e	An auxiliary symbol for defining a pseudo-metric on $\tilde{X}_{GG'}$	37
$\tilde{d}^\theta_{E'}$	A pseudo-metric on $\tilde{X}_{GG'}$	37
$F_{GG'}$	$E' \setminus E$	27
$F_{GG'}(vw, C)$	The set of those edges in $F_{GG'}$ tha cross the vw-cut C	42
f_v	The number of the video frame where vertex v occurs.	99
$G'(vw, C)$	$(V, F_{GG'}(vw, C) \cup C)$ the subgraph of G' that comprises all edges from $F_{GG'}(vw, C)$ and C	42
$G(v, C)$	Given cut C, the largest components of the graph $(V, E \setminus C)$ that contain v	42
$[k]$	The set of numbers $\{1, 2, \dots, k\}$.	68
lifted_costs$[u]$	The cost of the minimal ut-path w.r.t. to costs of all lifted edges connecting v with the vertices of the path.	87
$L_{\max}$	The maximal distance between two video frames such that edges are created between their detections.	99
M_G	The set of all multicuts of G	25
next$[u]$	The best neighbor of vertex u w.r.t. values in lifted_cost. That is, next$[u]$ = $\operatorname{argmin}_{w:uw\in\delta^+_E(u)}$ lifted_cost$[w]$	87
$\mathcal{R}_G$	Reachability relation in a directed graph G	55
$\mathcal{S}$	The set of subproblems in a Lagrange decomposition	83
$S_{GG'}(vw, C)$	The set of all decompositions of G into (vw, C)- connected components	42
V_i	The set of all vertices in the i-th time frame.	55
vw-cuts(G)	The set of all cuts between v and w in graph G	27
vw-paths(G)	The set of all paths from v to w in G	13
$\mathcal{X}^C$	Feasible set of subproblem for cut C	90
X_G	Set of characteristic functions of multicuts of G	7
$X_{GG'}$	Set of characteristic functions of multicuts of G' lifted from G	13
$X_{GG'}[\tilde{x}]$	Completions of $\tilde{x}$ in $X_{GG'}$	28

$\hat{X}_{GG'}$	The set of all maximally specific partial characterizations of multicuts of G' lifted from G	33
$\tilde{X}_{GG'}$	The set of all partial characterizations of multicuts of G' lifted from G	28
$\mathcal{X}^P$	Feasible set of subproblem for path P	89
$\mathcal{X}^{\mathsf{s}}$	The feasible set of subproblem $\mathsf{s} \in \mathcal{S}$	83
$\mathcal{X}^{out(v)}$	Feasible set of the outflow subproblem for node v	84
$Y^{DP}(G, s, t)$	Set of characteristic functions of node disjoint st-paths(G)	16, 17, 22, 54
$Y^{EDP}(G, s, t)$	Set of characteristic functions of edge disjoint st-paths(G)	16, 17

ACRONYMS

DFS	Depth First Search	87
DM	Deep Matching	74
DP	Disjoint Paths	22
GAEC	Greedy Additive Edge Contraction	10
ILP	Integer Linear Programming	iii
KL	Kernighan Lin	10
KLj	Kernighan Lin with joins	10
LAP	Linear Assignment Problem	90
LDP	Lifted Disjoint Paths	54
LMC	Lifted Multicut	13, 39
LP	Linear Programming	9
MC	Multicut	7, 38
MCF	Minimum Cost Flow	20
MLP	Multi-Layer Perceptron	117
MOT	Multiple Object Tracking	iii

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

BIBLIOGRAPHY

A. Abbas and P. Swoboda (2021). RAMA: A Rapid Multicut Algorithm on GPU, *arXiv preprint arXiv:2109.01838*. 10, 11

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin (1988). *Network flows*, Cambridge, Mass.: Alfred P. Sloan School of Management, Massachusetts. 16, 20, 21, 55, 90, 92

A. Alush and J. Goldberger (2012). Ensemble segmentation using efficient integer linear programming, *IEEE transactions on pattern analysis and machine intelligence*, vol. 34(10), pp. 1966–1977. 11

B. Andres, J. H. Kappes, T. Beier, U. Köthe, and F. A. Hamprecht (2011). Probabilistic image segmentation with closedness constraints, in *2011 International Conference on Computer Vision 2011*. 9, 11

B. Andres, T. Kroeger, K. L. Briggman, W. Denk, N. Korogod, G. Knott, U. Koethe, and F. A. Hamprecht (2012). Globally optimal closed-surface segmentation for connectomics, in *European Conference on Computer Vision 2012*. 9, 11, 12

B. Andres, J. Yarkony, B. Manjunath, S. Kirchhoff, E. Turetken, C. C. Fowlkes, and H. Pfister (2013). Segmenting planar superpixel adjacency graphs wrt non-planar superpixel affinity graphs, in *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition 2013*. 10, 12, 15

P. Arbeláez, M. Maire, C. C. Fowlkes, and J. Malik (2011). Contour Detection and Hierarchical Image Segmentation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33(5), pp. 898–916. 37

C. Arora and A. Globerson (2013). Higher Order Matching for Consistent Multiple Target Tracking, in *Proceedings of the IEEE International Conference on Computer Vision 2013*. 52

Y. Bachrach, P. Kohli, V. Kolmogorov, and M. Zadimoghaddam (2013). Optimal coalition structure generation in cooperative graph games, in *Twenty-Seventh AAAI Conference on Artificial Intelligence 2013*. 9

N. Bansal, A. Blum, and S. Chawla (2004). Correlation Clustering, *Machine Learning*, vol. 56(1–3), pp. 89–113. 8, 9

T. Beier, B. Andres, U. Köthe, and F. A. Hamprecht (2016). An efficient fusion move algorithm for the minimum cost lifted multicut problem, in *European Conference on Computer Vision 2016*. 14, 15

T. Beier, F. A. Hamprecht, and J. H. Kappes (2015). Fusion moves for correlation clustering, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2015*. 10, 11, 13, 14

T. Beier, T. Kroeger, J. H. Kappes, U. Kothe, and F. A. Hamprecht (2014). Cut, glue & cut: A fast, approximate solver for multicut partitioning, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2014*. 10, 11

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.* (2017). Multicut brings automated neurite segmentation closer to human performance, *Nature methods*, vol. 14(2), pp. 101–102. 15

J. Berclaz, F. Fleuret, E. Turetken, and P. Fua (2011). Multiple object tracking using k-shortest paths optimization, *IEEE transactions on pattern analysis and machine intelligence*, vol. 33(9), pp. 1806–1819. 22

S. Berg, D. Kutra, T. Kroeger, C. N. Straehle, B. X. Kausler, C. Haubold, M. Schiegg, J. Ales, T. Beier, M. Rudy, *et al.* (2019). Ilastik: interactive machine learning for (bio) image analysis, *Nature Methods*, vol. 16(12), pp. 1226–1232. 12

P. Bergmann, T. Meinhardt, and L. Leal-Taixe (2019). Tracking Without Bells and Whistles, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) 2019*. 73, 76, 77, 78, 79, 119

K. Bernardin and R. Stiefelhagen (2008a). Evaluating multiple object tracking performance: the CLEAR MOT metrics, *EURASIP Journal on Image and Video Processing*, vol. 2008, pp. 1–10. 75

K. Bernardin and R. Stiefelhagen (2008b). Evaluating multiple object tracking performance: The CLEAR MOT metrics, *EURASIP Journal on Image and Video Processing*, vol. 2008. 120

U. Brandes and D. Wagner (2000). A linear time algorithm for the arc disjoint Menger problem in planar directed graphs, *Algorithmica*, vol. 28(1), pp. 16–36. 17

G. Braso and L. Leal-Taixe (2020). Learning a Neural Solver for Multiple Object Tracking, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2020*. 22, 119

W. Brendel, M. Amer, and S. Todorovic (2011). Multiobject tracking as maximum weight independent set, in *IEEE Conference on Computer Vision and Pattern Recognition 2011*. 52

R. G. Busacker and P. J. Gowen (1960). A procedure for determining a family of minimum-cost network flow patterns, Technical report, RESEARCH ANALYSIS CORP MCLEAN VA. 21

V. Chari, S. Lacoste-Julien, I. Laptev, and J. Sivic (2015). On pairwise costs for network flow multi-object tracking, in *IEEE Conference on Computer Vision and Pattern Recognition 2015*. 52

M. Charikar, V. Guruswami, and A. Wirth (2005). Clustering with qualitative information, *Journal of Computer and System Sciences*, vol. 71(3), pp. 360–383. 9

L. Chen, H. Ai, R. Chen, and Z. Zhuang (2019). Aggregate Tracklet Appearance Features for Multi-Object Tracking, *IEEE Signal Processing Letters*, vol. 26(11), pp. 1613–1617. 77

L. Chen, H. Ai, C. Shang, Z. Zhuang, and B. Bai (2017). Online multi-object tracking with convolutional neural networks, in *IEEE International Conference on Image Processing 2017*. 77

S. Chopra and M. Rao (1993). The partition problem, *Mathematical Programming*, vol. 59(1–3), pp. 87–115. 2, 7, 25

P. Chu, H. Fan, C. C. Tan, and H. Ling (2019). Online multi-object tracking with instance-aware tracker and dynamic model refreshment, in *IEEE Winter Conference on Applications of Computer Vision 2019*. 77

P. Chu and H. Ling (2019). Famnet: Joint learning of feature, affinity and multi-dimensional assignment for online multiple object tracking, in *IEEE International Conference on Computer Vision 2019*. 77

S. A. Cook (1971). The complexity of theorem-proving procedures, in *Proceedings of the third annual ACM symposium on Theory of computing 1971*. 67, 70

G. Dantzig (2016). *Linear programming and extensions*, Princeton university press. 22

G. Dantzig and D. Fulkerson (1955). ON THE MAX FLOW MIN CUT THEOREM OF NETWORKS, Technical report, RAND CORP SANTA MONICA CA. 4, 16

G. B. Dantzig and D. Fulkerson (1956). On the min cut max flow theorem of networks, *Annals of Mathematical Study*, vol. 38, pp. 215–222. 17

I. Davidson and S. Ravi (2005). Clustering with constraints: Feasibility issues and the k-means algorithm, in *Proceedings of the 2005 SIAM international conference on data mining 2005*. 24

I. Davidson and S. Ravi (2007). Intractability and clustering with constraints, in *Proceedings of the 24th international conference on Machine learning 2007*. 24

A. Dehghan, S. Modiri Assari, and M. Shah (2015). GMMCP tracker: Globally optimal generalized maximum multi clique problem for multiple object tracking, in *IEEE Conference on Computer Vision and Pattern Recognition 2015*. 52, 53

E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica (2006). Correlation clustering in general weighted graphs, *Theoretical Computer Science*, vol. 361(2–3), pp. 172–187. 8, 9

P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé (2020). MOT20: A benchmark for multi object tracking in crowded scenes, *arXiv:2003.09003[cs]*. 5, 83, 120, 124, 128, 135

M. M. Deza, M. Grötschel, and M. Laurent (1991). Complete descriptions of small multicut polytopes, in *Applied Geometry and Discrete Mathematics – The Victor Klee Festschrift, volume 4 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science 1991*. 2, 8

M. M. Deza, M. Grötschel, and M. Laurent (1992). Clique-Web Facets for Multicut Polytopes, *Mathematics of Operations Research*, vol. 17(4), pp. 981–1000. 8, 10

M. M. Deza and M. Laurent (1997). *Geometry of cuts and metrics*, vol. 2, Springer. 8, 39

K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian (2019). Centernet: Keypoint triplets for object detection, in *Proceedings of the IEEE/CVF International Conference on Computer Vision 2019*. 51

J. Edmonds and R. M. Karp (1972). Theoretical improvements in algorithmic efficiency for network flow problems, *Journal of the ACM (JACM)*, vol. 19(2), pp. 248–264. 21

T. Eilam-Tzoreff (1998). The disjoint shortest paths problem, *Discrete applied mathematics*, vol. 85(2), pp. 113–138. 20

S. Even, A. Itai, and A. Shamir (1975). On the complexity of time table and multi-commodity flow problems, in *16th Annual Symposium on Foundations of Computer Science (sfcs 1975) 1975*. 19, 67, 68, 69

S. Even and R. E. Tarjan (1975). Network flow and testing graph connectivity, *SIAM journal on computing*, vol. 4(4), pp. 507–518. 17

T. Feder and R. Motwani (1991). Clique partitions, graph compression and speeding-up algorithms, in *Proceedings of the twenty-third annual ACM symposium on Theory of computing 1991*. 17

D. H. Fisher (1987). Knowledge acquisition via incremental conceptual clustering, *Machine learning*, vol. 2(2), pp. 139–172. 24

L. R. Ford and D. R. Fulkerson (1956). Maximal flow through a network, *Canadian journal of Mathematics*, vol. 8, pp. 399–404. 16, 17

L. R. Ford and D. R. Fulkerson (1957). A simple algorithm for finding maximal network flows and an application to the Hitchcock problem, *Canadian journal of Mathematics*, vol. 9, pp. 210–218. 17

L. R. Ford Jr (1956). Network flow theory, Technical report, Rand Corp Santa Monica Ca. 16

S. Fortune, J. Hopcroft, and J. Wyllie (1980). The directed subgraph homeomorphism problem, *Theoretical Computer Science*, vol. 10(2), pp. 111–121. 19, 20

R. Ganian, T. Hamm, and S. Ordyniak (2021). The Complexity of Object Association in Multiple Object Tracking, in *Proceedings of the AAAI Conference on Artificial Intelligence 2021*. 82

N. Garg, V. V. Vazirani, and M. Yannakakis (1997). Primal-dual approximation algorithms for integral flow and multicut in trees, *Algorithmica*, vol. 18(1), pp. 3–20. 8

A. V. Goldberg (1997). An efficient implementation of a scaling minimum-cost flow algorithm, *Journal of algorithms*, vol. 22(1), pp. 1–29. 22

M. Grötschel and Y. Wakabayashi (1989). A cutting plane algorithm for a clustering problem, *Mathematical Programming*, vol. 45(1), pp. 59–96. 7, 9, 12

M. Grötschel and Y. Wakabayashi (1990). Facets of the clique partitioning polytope, *Mathematical Programming*, vol. 47(1), pp. 367–387. 7

T. Grünwald (1938). Ein neuer Beweis eines mengerschen Satzes, *Journal of the London Mathematical Society*, vol. 1(3), pp. 188–192. 17

M. Guignard and S. Kim (1987). Lagrangean decomposition for integer programming: theory and applications, *RAIRO-Operations Research-Recherche Opérationnelle*, vol. 21(4), pp. 307–323. 83

L. Gurobi Optimization (2019). *Gurobi Optimizer Reference Manual*. 4, 66, 81, 128

R. W. Hamming (1950). Error detecting and error correcting codes, *The Bell system technical journal*, vol. 29(2), pp. 147–160. 3, 24

R. Henschel, L. Leal-Taixé, D. Cremers, and B. Rosenhahn (2018). Fusion of Head and Full-Body Detectors for Multi-Object Tracking, in *IEEE Conference on Computer Vision and Pattern Recognition Workshops 2018*. 53, 74, 79

R. Henschel, L. Leal-Taixé, and B. Rosenhahn (2014). Efficient multiple people tracking using minimum cost arborescences, in *German Conference on Pattern Recognition 2014*. 52

R. Henschel, L. Leal-Taixé, B. Rosenhahn, and K. Schindler (2016). Tracking with multi-level features, *arXiv preprint arXiv:1607.07304*. 53

R. Henschel, T. von Marcard, and B. Rosenhahn (2019a). Simultaneous Identification and Tracking of Multiple People Using Video and IMUs, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops 2019*. 53

R. Henschel, Y. Zou, and B. Rosenhahn (2019b). Multiple people tracking using body and joint detections, in *IEEE Conference on Computer Vision and Pattern Recognition Workshops 2019*. 74, 77

K. Ho, A. Chatzimichailidis, M. Keuper, and J. Keuper (2021). MSM: Multi-stage Multicuts for Scalable Image Clustering, in *High Performance Computing 2021*. 11, 13

K. Ho, A. Kardoost, F.-J. Pfreundt, J. Keuper, and M. Keuper (2020). A Two-Stage Minimum Cost Multicut Approach to Self-Supervised Multiple Person Tracking, in *Proceedings of the Asian Conference on Computer Vision (ACCV) 2020*. 15

A. Hoffman and J. Kruskal (1956). Integral boundary points of convex polyhedra, in “Linear Inequalities and Related Systems”(HW Kuhn and AW Tucker, Eds.), *Annals of Mathematical Studies*, (38). 21

M. Hofmann, D. Wolf, and G. Rigoll (2013). Hypergraphs for joint multi-view reconstruction and multi-object tracking, in *IEEE Conference on Computer Vision and Pattern Recognition 2013*. 52

A. Horňáková, R. Henschel, B. Rosenhahn, and P. Swoboda (2020). Lifted Disjoint Paths with Application in Multiple Object Tracking, in *Proceedings of the 37th International Conference on Machine Learning (ICML 2020) 2020*. 3, 51, 71, 74, 75, 82

A. Hornakova, T. Kaiser, P. Swoboda, M. Rolinek, B. Rosenhahn, and R. Henschel (2021). Making Higher Order MOT Scalable: An Efficient Approximate Solver for Lifted Disjoint Paths, in *Proceedings of the IEEE/CVF International Conference on Computer Vision 2021*. 4, 118, 120

A. Horňáková, J.-H. Lange, and B. Andres (2017). Analysis and Optimization of Graph Decompositions by Lifted Multicuts, in *International Conference on Machine Learning 2017*. 2, 3, 13, 14, 23, 24, 42

T. C. Hu (1969). Integer programming and network flows, Technical report, WISCONSIN UNIV MADISON DEPT OF COMPUTER SCIENCES. 8

W. Hu, X. Shi, Z. Zhou, J. Xing, H. Ling, and S. Maybank (2019). Dual L1-Normalized Context Aware Tensor Power Iteration and Its Applications to Multi-object Tracking and Multi-graph Matching, *International Journal of Computer Vision*. 52, 53

T. Ibaraki and S. Poljak (1991). Weak Three-Linking in Eulerian Dgraphs, *SIAM journal on Discrete Mathematics*, vol. 4(1), pp. 84–98. 20

E. Insafutdinov, M. Andriluka, L. Pishchulin, S. Tang, E. Levinkov, B. Andres, and B. Schiele (2017). ArtTrack: Articulated Multi-Person Tracking in the Wild, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017*. 12

E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele (2016). Deepercut: A deeper, stronger, and faster multi-person pose estimation model, in *European Conference on Computer Vision 2016*. 9, 12

M. Iri (1960). A new method of solving transportation-network problems, *Journal of the Operations Research Society of Japan*, vol. 3(1), p. 2. 21

W. S. Jewell (1958). Optimal flow through networks, in *Operations Research 1958*. 21

H. Jiang, S. Fels, and J. J. Little (2007). A linear programming approach for multiple object tracking, in *2007 IEEE Conference on Computer Vision and Pattern Recognition 2007*. 52

J. H. Kappes, B. Andres, F. A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, T. Kröger, J. Lellmann, *et al.* (2015a). A comparative study of modern inference techniques for structured discrete energy minimization problems, *International Journal of Computer Vision*, vol. 115(2), pp. 155–184. 10, 82

J. H. Kappes, M. Speth, B. Andres, G. Reinelt, and C. Schn (2011). Globally optimal image partitioning by multicuts, in *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition 2011*. 10, 11, 12

J. H. Kappes, M. Speth, G. Reinelt, and C. Schnörr (2016). Higher-order segmentation via multicuts, *Computer Vision and Image Understanding*, vol. 143, pp. 104–119. 10, 11

J. H. Kappes, P. Swoboda, B. Savchynskyy, T. Hazan, and C. Schnörr (2015b). Probabilistic correlation clustering and image partitioning using perturbed multicuts, in *International Conference on Scale Space and Variational Methods in Computer Vision 2015*. 10, 11

A. Kardoost and M. Keuper (2019). Solving Minimum Cost Lifted Multicut Problems by Node Agglomeration, in *Computer Vision – ACCV 2018 2019*. 14, 15

A. Kardoost and M. Keuper (2021). Uncertainty in Minimum Cost Multicuts for Image and Motion Segmentation, in *Uncertainty in Artificial Intelligence (UAI) 2021*. 11, 12

R. M. Karp (1975). On the computational complexity of combinatorial problems, *Networks*, vol. 5(1), pp. 45–68. 19

A. V. Karzanov (1973). On finding maximum flows in networks with special structure and some applications, *Matematicheskie Voprosy Upravleniya Proizvodstvom*, vol. 5, pp. 81–94. 17

B. W. Kernighan and S. Lin (1970). An efficient heuristic procedure for partitioning graphs, *The Bell system technical journal*, vol. 49(2), pp. 291–307. 10

M. Keuper (2017). Higher-Order Minimum Cost Lifted Multicuts for Motion Segmentation, in *2017 IEEE International Conference on Computer Vision (ICCV) 2017*. 14, 15

M. Keuper, B. Andres, and T. Brox (2015a). Motion Trajectory Segmentation via Minimum Cost Multicuts, in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) 2015*. 10, 12, 15

M. Keuper, E. Levinkov, N. Bonneel, G. Lavoué, T. Brox, and B. Andres (2015b). Efficient decomposition of image and mesh graphs by lifted multicuts, in *Proceedings of the IEEE International Conference on Computer Vision 2015*. 10, 13, 14, 15

M. Keuper, S. Tang, B. Andres, T. Brox, and B. Schiele (2018). Motion segmentation & multiple object tracking by correlation co-clustering, *IEEE transactions on pattern analysis and machine intelligence*, vol. 42(1), pp. 140–153. 12, 14

S. Kim, S. Nowozin, P. Kohli, and C. Yoo (2011). Higher-order correlation clustering for image segmentation, *Advances in neural information processing systems*, vol. 24, pp. 1530–1538. 10, 11

S. Kim, C. D. Yoo, S. Nowozin, and P. Kohli (2014). Image Segmentation UsingHigher-Order Correlation Clustering, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36(9), pp. 1761–1774. 10, 11

A. Kirillov, E. Levinkov, B. Andres, B. Savchynskyy, and C. Rother (2017). InstanceCut: From Edges to Instances With MultiCut, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017*. 12

V. Kolmogorov (2006). Convergent tree reweighted message passing for energy minimization, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28(10). 82

V. Kolmogorov (2014). A new look at reweighted message passing, *IEEE transactions on pattern analysis and machine intelligence*, vol. 37(5), pp. 919–930. 82

A. Kotzig (1956). *Súvislost' a pravidelná súvislost' konečných grafov*. 17

P. Kovács (2015). Minimum-cost flow algorithms: an experimental evaluation, *Optimization Methods and Software*, vol. 30(1), pp. 94–127. 3, 21

T. Kroeger, S. Mikula, W. Denk, U. Koethe, and F. A. Hamprecht (2013). Learning to segment neurons with non-local quality measures, in *International Conference on Medical Image Computing and Computer-Assisted Intervention 2013*. 11

J.-H. Lange (2020). Multicut Optimization Guarantees & Geometry of Lifted Multicuts. 8

J.-H. Lange and B. Andres (2020). On the lifted multicut polytope for trees, in *DAGM German Conference on Pattern Recognition 2020*. 14

J.-H. Lange, B. Andres, and P. Swoboda (2019). Combinatorial persistency criteria for multicut and max-cut, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2019*. 11

J.-H. Lange, A. Karrenbauer, and B. Andres (2018). Partial optimality and fast lower bounds for weighted correlation clustering, in *International Conference on Machine Learning 2018*. 11

J.-H. Lange and P. Swoboda (2021). Efficient Message Passing for 0–1 ILPs with Binary Decision Diagrams, in *Proceedings of the 38th International Conference on Machine Learning 2021*. 82

A. S. LaPaugh and R. L. Rivest (1980). The subgraph homeomorphism problem, *Journal of Computer and System Sciences*, vol. 20(2), pp. 133–149. 18, 20

L. Leal-Taixé, M. Fenzi, A. Kuznetsova, B. Rosenhahn, and S. Savarese (2014). Learning an image-based motion context for multiple people tracking, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2014*. 22

L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler (2015). MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking, *arXiv:1504.01942 [cs]*. 3, 5, 74, 82, 120

L. Leal-Taixé, G. Pons-Moll, and B. Rosenhahn (2011). Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker, in *2011 IEEE international conference on computer vision workshops (ICCV workshops) 2011*. 22

L. Leal-Taixé, G. Pons-Moll, and B. Rosenhahn (2012). Branch-and-price global optimization for multi-view multi-target tracking, in *IEEE Conference on Computer Vision and Pattern Recognition 2012*. 52

E. Levinkov, A. Kirillov, and B. Andres (2017a). A Comparative Study of Local Search Algorithms for Correlation Clustering, in *Pattern Recognition 2017*. 10

E. Levinkov, J. Tompkin, N. Bonneel, S. Kirchhoff, B. Andres, and H. Pfister (2016). Interactive multicut video segmentation, in *Pacific Graphics 2016*. 10, 12

E. Levinkov, J. Uhrig, S. Tang, M. Omran, E. Insafutdinov, A. Kirillov, C. Rother, T. Brox, B. Schiele, and B. Andres (2017b). Joint Graph Decomposition and Node Labeling: Problem, Algorithms, Applications, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017*. 14, 15

Y. Li, C. Huang, and R. Nevatia (2009). Learning to associate: Hybridboosted multi-target tracker for crowded scene, in *IEEE Conference on Computer Vision and Pattern Recognition 2009*. 75

T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar (2018). Focal Loss for Dense Object Detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, pp. 1–1. 118

Q. Liu, Q. Chu, B. Liu, and N. Yu (2020). GSM: Graph Similarity Model for Multi-Object Tracking, in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20 2020*. 119

L. Ma, S. Tang, M. J. Black, and L. Van Gool (2018). Customized multi-person tracker, in *Asian Conference on Computer Vision 2018*. 77

Y. Ma, Q. Yu, and I. Cohen (2009). Target tracking with incomplete detection, *Computer Vision and Image Understanding*, vol. 113(4), pp. 580–587. 22

A. L. Maas, A. Y. Hannun, and A. Y. Ng (2013). Rectifier Nonlinearities Improve Neural Network Acoustic Models, in *Proceedings of the International Conference on Machine Learning 2013*. 117

M. Meilă (2007). Comparing clusterings—an information based distance, *Journal of Multivariate Analysis*, vol. 98(5), pp. 873–895. 24

K. Menger (1927). Zur allgemeinen kurventheorie, *Fundamenta Mathematicae*, vol. 10(1), pp. 96–115. 17

A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler (2016). MOT16: A Benchmark for Multi-Object Tracking, *arXiv:1603.00831 [cs]*. 3, 5, 73, 74, 82, 120, 124, 135

H. Nagamochi and T. Ibaraki (1992). A linear-time algorithm for finding a sparse k-connected spanning subgraph of ak-connected graph, *Algorithmica*, vol. 7(1), pp. 583–596. 17

S. Nowozin and S. Jegelka (2009). Solution stability in linear programming relaxations: Graph partitioning and unsupervised learning, in *Proceedings of the 26th Annual International Conference on Machine Learning 2009*. 9, 10

S. Nowozin and C. H. Lampert (2010). Global interactions in random field models: A potential function ensuring connectedness, *SIAM Journal on Imaging Sciences*, vol. 3(4), pp. 1048–1074. 13

M. Oellrich (2008). Minimum Cost Disjoint Paths under Arc Dependences. Algorithms for Practice. 18, 19

C. Pape, T. Beier, P. Li, V. Jain, D. D. Bock, and A. Kreshuk (2017). Solving large multicut problems for connectomics via domain decomposition, in *Proceedings of the IEEE International Conference on Computer Vision Workshops 2017*. 11, 15

C. Pape, A. Matskevych, A. Wolny, J. Hennies, G. Mizzon, M. Louveaux, J. Musser, A. Maizel, D. Arendt, and A. Kreshuk (2019). Leveraging domain knowledge to improve microscopy image segmentation with lifted multicuts, *Frontiers in Computer Science*, vol. 1, p. 6. 15

H. Pirsiavash, D. Ramanan, and C. C. Fowlkes (2011). Globally-optimal greedy algorithms for tracking a variable number of objects, in *CVPR 2011 2011*. 22

L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. V. Gehler, and B. Schiele (2016). DeepCut: Joint Subset Partition and Labeling for Multi Person Pose Estimation, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2016*. 9, 12

W. M. Rand (1971). Objective Criteria for the Evaluation of Clustering Methods, *Journal of the American Statistical Association*, vol. 66(336), pp. 846–850. 3, 24, 36

J. Redmon, S. Divvala, R. Girshick, and A. Farhadi (2016). You only look once: Unified, real-time object detection, in *Proceedings of the IEEE conference on computer vision and pattern recognition 2016*. 51

S. Ren, K. He, R. Girshick, and J. Sun (2015). Faster r-cnn: Towards real-time object detection with region proposal networks, *arXiv preprint arXiv:1506.01497*. 51

H. Ripphausen-Lipa, D. Wagner, and K. Weihe (1997). The vertex-disjoint Menger problem in planar graphs, *SIAM Journal on Computing*, vol. 26(2), pp. 331–349. 17

E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi (2016a). Performance Measures and a Data Set for Multi-target, Multi-camera Tracking, in *Computer Vision – ECCV 2016 Workshops 2016*. 120

E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, and C. Tomasi (2016b). Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking, in *European Conference on Computer Vision Workshop on Benchmarking Multi-Target Tracking 2016*. 73, 75, 117

E. Ristani and C. Tomasi (2014). Tracking multiple people online and in real time, in *Asian Conference on Computer Vision 2014*. 12

E. Ristani and C. Tomasi (2018). Features for multi-target multi-camera tracking and re-identification, in *IEEE Conference on Computer Vision and Pattern Recognition 2018*. 74, 79

M. Rolínek, P. Swoboda, D. Zietlow, A. Paulus, V. Musil, and G. Martius (2020). Deep graph matching via blackbox differentiation of combinatorial solvers, in *European Conference on Computer Vision 2020*. 82

A. Sadeghian, A. Alahi, and S. Savarese (2017). Tracking the untrackable: Learning to track multiple cues with long-term dependencies, in *IEEE International Conference on Computer Vision 2017*. 52, 77

B. Savchynskyy *et al.* (2019). Discrete graphical models—an optimization perspective, *Foundations and Trends® in Computer Graphics and Vision*, vol. 11(3-4), pp. 160–429. 82

A. Schrijver (1994). Finding k disjoint paths in a directed planar graph, *SIAM Journal on Computing*, vol. 23(4), pp. 780–788. 20

A. Schrijver (2003). *Combinatorial optimization: polyhedra and efficiency*, vol. 24, Springer Science & Business Media. 8, 16, 17, 20

H. Sheng, Y. Zhang, J. Chen, Z. Xiong, and J. Zhang (2018). Heterogeneous association graph fusion for target association in multiple object tracking, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29(11), pp. 3269–3280. 77

J. Song, B. Andres, M. J. Black, O. Hilliges, and S. Tang (2019). End-to-End Learning for Graph Decomposition, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) 2019*. 12, 13, 14

J. W. Suurballe (1974). Disjoint paths in a network, *Networks*, vol. 4(2), pp. 125–145. 21, 22

P. Swoboda and B. Andres (2017). A message passing algorithm for the minimum cost multicut problem, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2017*. 10, 82

P. Swoboda, D. Kainm"uller, A. Mokarian, C. Theobalt, and F. Bernard (2019). A Convex Relaxation for Multi-Graph Matching, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2019*. 82

P. Swoboda, J. Kuske, and B. Savchynskyy (2017a). A Dual Ascent Framework for Lagrangean Decomposition of Combinatorial Problems, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017*. 5, 10, 81, 82, 83, 84

P. Swoboda, C. Rother, H. Abu Alhaija, D. Kainmuller, and B. Savchynskyy (2017b). A Study of Lagrangean Decompositions and Dual Ascent Solvers for Graph Matching, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017*. 82

S. Tang, B. Andres, M. Andriluka, and B. Schiele (2015). Subgraph Decomposition for Multi-Target Tracking, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2015*. 10, 12

S. Tang, B. Andres, M. Andriluka, and B. Schiele (2016). Multi-person Tracking by Multicut and Deep Matching, in *Computer Vision – ECCV 2016 Workshops 2016*. 10, 12, 14, 74

S. Tang, M. Andriluka, B. Andres, and B. Schiele (2017). Multiple people tracking by lifted multicut and person re-identification, in *Proceedings of the IEEE conference on computer vision and pattern recognition 2017*. 14, 15, 53, 55, 79

R. E. Tarjan (1974). Testing graph connectivity, in *Proceedings of the sixth annual ACM symposium on Theory of computing 1974*. 17

T. Tholey (2012). Linear time algorithms for two disjoint paths problems on directed acyclic graphs, *Theoretical Computer Science*, vol. 465, pp. 35–48. 19

N. Tomizawa (1971). On some techniques useful for solution of transportation network problems, *Networks*, vol. 1(2), pp. 173–194. 21

V. V. Vazirani (2001). *Approximation algorithms*, vol. 1, Springer. 8

T. Voice, M. Polukarov, and N. R. Jennings (2012). Coalition Structure Generation over Graphs, *Journal of Artificial Intelligence Research*, vol. 45(1), pp. 165–196. 9

T. von Marcard, R. Henschel, M. J. Black, B. Rosenhahn, and G. Pons-Moll (2018). Recovering accurate 4d human pose in the wild using imus and a moving camera, in *Proceedings of the European Conference on Computer Vision (ECCV) 2018*. 53

J. Vygen (1994). *Disjoint paths*, Citeseer. 18

J. Vygen (1995). NP-completeness of some edge-disjoint paths problems, *Discrete Applied Mathematics*, vol. 61(1), pp. 83–90. 19

K. Wagstaff and C. Cardie (2000). Clustering with Instance-level Constraints, in *Proceedings of the Seventeenth International Conference on Machine Learning 2000*. 24

C. Wang, Y. Wang, Y. Wang, C.-T. Wu, and G. Yu (2019a). muSSP: Efficient Min-cost Flow Algorithm for Multi-object Tracking, in *Advances in Neural Information Processing Systems 2019*. 22

G. Wang, Y. Wang, H. Zhang, R. Gu, and J.-N. Hwang (2019b). Exploit the connectivity: Multi-object tracking with trackletnet, in *ACM International Conference on Multimedia 2019*. 77

L. Wei, S. Zhang, W. Gao, and Q. Tian (2018). Person transfer gan to bridge domain gap for person re-identification, in *IEEE Conference on Computer Vision and Pattern Recognition 2018*. 73, 117

K. Weihe (1997). Edge-disjoint (s, t)-paths in undirected planar graphs in linear time, *Journal of Algorithms*, vol. 23(1), pp. 121–138. 17

P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid (2013). DeepFlow: Large displacement optical flow with deep matching, in *IEEE Intenational Conference on Computer Vision 2013*. 74

T. Werner (2007). A linear programming approach to max-sum problem: A review, *IEEE transactions on pattern analysis and machine intelligence*, vol. 29(7), pp. 1165–1179. 82

T. Werner (2009). Revisiting the linear programming relaxation approach to Gibbs energy minimization and weighted constraint satisfaction, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32(8), pp. 1474–1488. 82

N. Wojke, A. Bewley, and D. Paulus (2017). Simple Online and Realtime Tracking with a Deep Association Metric, in *2017 IEEE International Conference on Image Processing (ICIP) 2017*. 52

A. Wolny, L. Cerrone, A. Vijayan, R. Tofanelli, A. V. Barro, M. Louveaux, C. Wenzl, S. Strauss, D. Wilson-Sánchez, R. Lymbouridou, *et al.* (2020). Accurate and versatile 3D segmentation of plant tissues at cellular resolution, *Elife*, vol. 9, p. e57613. 11

J. Xu, Y. Cao, Z. Zhang, and H. Hu (2019). Spatial-temporal relation networks for multi-object tracking, in *IEEE International Conference on Computer Vision 2019*. 52, 77

F. Yang, W. Choi, and Y. Lin (2016). Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers, in *Proceedings of the IEEE conference on computer vision and pattern recognition 2016*. 51

J. Yarkony, T. Beier, P. Baldi, and F. A. Hamprecht (2014). Parallel multicut segmentation via dual decomposition, in *International Workshop on New Frontiers in Mining Complex Patterns 2014*. 10, 11

J. Yarkony, A. Ihler, and C. C. Fowlkes (2012). Fast planar correlation clustering for image segmentation, in *European Conference on Computer Vision 2012*. 10, 11

A. R. Zamir, A. Dehghan, and M. Shah (2012). GMCP-tracker: Global multi-object tracking using generalized minimum clique graphs, in *European Conference on Computer Vision 2012*. 52, 53

C. Zhang, J. Yarkony, and F. A. Hamprecht (2014). Cell detection and segmentation using correlation clustering, in *International Conference on Medical Image Computing and Computer-Assisted Intervention 2014*. 11

L. Zhang, Y. Li, and R. Nevatia (2008). Global data association for multi-object tracking using network flows, in *IEEE Conference on Computer Vision and Pattern Recognition 2008*. 21, 22, 55

L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian (2015). Scalable person re-identification: A benchmark, in *IEEE International Conference on Computer Vision 2015*. 73, 117

Z. Zheng, X. Yang, Z. Yu, L. Zheng, Y. Yang, and J. Kautz (2019). Joint discriminative and generative learning for person re-identification, in *IEEE Conference on Computer Vision and Pattern Recognition 2019*. 73, 117

X. Zhou, V. Koltun, and P. Krähenbühl (2020). Tracking objects as points, in *European Conference on Computer Vision 2020*. 119

J. Zhu, H. Yang, N. Liu, M. Kim, W. Zhang, and M.-H. Yang (2018). Online multi-object tracking with dual matching attention networks, in *European Conference on Computer Vision 2018*. 52