Solution methods for partial inverse combinatorial optimization problems in which weights can only be increased

Eva Ley and Maximilian Merkert

TU Braunschweig
Institute for Mathematical Optimization
Universitätsplatz 2, 38106 Braunschweig, Germany
eva.ley@tu-braunschweig.de, m.merkert@tu-braunschweig.de

February 13, 2024

Abstract

Partial inverse combinatorial optimization problems are bilevel optimization problems in which the leader aims to incentivize the follower to include a given set of elements in the solution of their combinatorial problem. If the set of required elements defines a complete follower solution, the inverse combinatorial problem is solvable in polynomial time as soon as this holds true for the follower problem. In contrast, the same is not necessarily true for partial inverse problems, e.g. the partial inverse min cut problem has been shown to be NP-complete. In this paper, we consider partial inverse combinatorial optimization problems in which weights can only be increased. Furthermore, we assume that the lower-level combinatorial problem can be solved as a linear program. In this setting of allowing only weight increases, we show that the partial inverse shortest path problem on a directed acyclic graph is NP-complete even if only a single arc is required to be part of a shortest path. Moreover, the partial inverse assignment problem with a single required edge is NP-complete. For solving partial inverse combinatorial optimization problems with only weight increases, we present a novel branch-and-bound scheme that exploits the difference in complexity between complete inverse and partial inverse versions of a problem. For both primal heuristics and node relaxations, our scheme uses auxiliary problems that are basically complete inverse problems on similar instances, while branching is done on follower variables. We test our approach on partial inverse shortest path, assignment and min cut problems, and computationally compare it to an MPCC reformulation as well as a decomposition scheme.

Keywords: Bilevel Optimization, Inverse Problems, Mixed-Integer Programming, Branch and Bound

Mathematics Subject Classification: 90C11, 90C26, 90C27, 90C60, 91A65

1 Introduction

A partial inverse combinatorial problem (PICP) is a bilevel optimization problem in which the leader aims to find minimal-weight modifications for the objective of a follower's combinatorial problem such that a lower-level optimal solution will be chosen that extends a given partial lower-level solution. It is therefore a bilevel optimization problem with bound constraints on lower-level variables on the upper level. Such

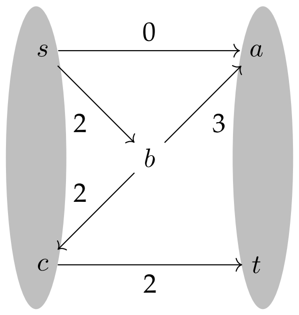

Figure 1: Completing partial min cut by min cut after contraction of required parts is not optimal, as demonstrated in this example instance inspired by [15]: Require the partial cut $\{s, c\}, \{a, t\}$. The contracted graph with vertices $\{s, c\}, \{a, t\}, b$ allows two cuts $I = (\{s, c, b\}, \{a, t\})$ with value 5 or $J = (\{s, c\}, \{a, t, b\})$ with value 4. However, reducing capacities for I by 3 (reduce capacity of arc ba to 0) is better than decreasing weights for J by 4 (reduce capacity of arcs sb, ct both by 2 to 0).

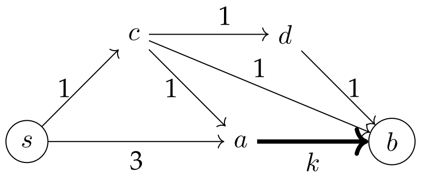

Figure 2: Shortest path completion is not optimal: Let $k \geq 3$. The shortest s-a-path is via vertex c. The inverse shortest path problem for edges $\{sc, ca, ab\}$ adds weight k to arc cb and $k-1$ to cd or db, so in total $2k-1$. In contrast, only a total weight of $k+1$ is added to arc sc for the complete path $\{sa, ab\}$.

problems arise e.g. in economic applications in which a planner wants rational followers to act in a desired way, see e.g. [10]. A PICP can also model parameter estimation problems from data science, in which the goal is to find reasonable values for a set of unknown parameters that explain partially observable follower behavior, see e.g. [24].

A *complete* inverse combinatorial problem (ICP), i.e. an ICP for which the predefined partial lower-level solution in fact specifies a complete solution, is solvable in polynomial time as soon as the lower-level problem is solvable in polynomial time [2]. In contrast, a *partial* inverse combinatorial problem can be $\mathcal{NP}$-hard even if the lower-level problem is an 'easy' combinatorial optimization problem solvable in polynomial time. Hence, the main difficulty of a PICP can be seen in determining which lower-level feasible solution will turn out to be optimal in an optimal solution of the PICP. Note that the number of candidate lower-level feasible solutions can be exponential. It is natural to guess that this lower-level solution will be a minimal (complete) lower-level solution that extends the partial solution. However, this is not guaranteed in general. For example, the partial inverse min cut problem (PIMCP) cannot be solved by the method of first computing a lower-level completion and then solving an inverse min cut problem, see Figure 1 for a small example. In fact, the PIMCP is $\mathcal{NP}$-complete [15]. However, some PICPs can be solved via minimal lower-level completions in polynomial time, for example the partial inverse shortest path problem (PISPP) on directed acyclic graphs (DAGs), see Theorem 2.9, or the partial inverse assignment problem (PIAP) [29]. This, however, may change again in the presence of additional constraints, e.g. bounds on weight modifications, see Figure 2 for an illustrating example on the PISPP on DAGs with only weight increases; an example for PIAP can be found in [28]. A solution based on a minimal completion can have a value with an arbitrarily large gap to the optimal value, as the example in Figure 2 shows.

The main contribution of this paper is a novel branch-and-bound method for a certain class of $\mathcal{NP}$-complete PICPs in which weights can only be increased – which we denote by PICP-W+. The main idea is to exploit the above-mentioned fact that many such problems are solvable in polynomial time as soon as we consider *complete* instead of *partial* versions of the respective ICPs. We will heavily make use of this for both the primal and dual bounding procedure. Throughout this work, we use shortest path problems on a directed acyclic graph, assignment problems and min cut problems as example cases. Despite examples like the ones in Figure 2, it turns out that solutions based on minimal completions can be used to obtain strong primal bounds in all these cases. For obtaining dual bounds we propose a construction that yields a problem sharing essential similarities with a *complete* ICP and which can be explicitly modelled as an ICP for two out of three of the example cases. This construction requires valid and strong dual bounds on the lower-level objective in an optimal solution, which are efficiently obtainable if weights can only be increased. In a computational study on the example problem classes, we show that the resulting algorithm is already competitive even without any other, classical, relaxation techniques or primal heuristics, when compared to a state-of-the-art solver applied to a single-level reformulation. In addition, we fill some gaps in the complexity landscape of the problems considered, by giving proofs for the $\mathcal{NP}$-completeness of PISPP-W+ on a directed acyclic graph and the PIAP-W+, in both cases with a single required arc. Instances from both problem classes are relevant for the computational study.

In this work, we will use the *optimistic assumption*, i.e. if the follower is indifferent towards two or more solutions, they will choose the one most favourable to the leader. For optimistic PICPs, the existence of one lower-level optimal solution fulfilling the upper level's requirements suffices. In contrast, for PICPs with a pessimistic stance every lower-level optimal solution must fulfil the upper-level's requirements. For PICPs, however, solving the optimistic or the pessimistic version is essentially the same from an algorithmic point of view, see Section 2.6.

In the following Section 2, we formally define PICPs, give optimality conditions and describe PICPs that we use in the following. Furthermore, we provide a literature review, present general solution approaches and briefly discuss the pessimistic version of PICPs. Afterwards in Section 3, we prove two complexity results on PISPP-W+ and PIAP-W+, i.e. PISPP respectively PIAP with only weight increases. In Section 4, we present our branch-and-bound method for PICP-W+. The main emphasis is on primal and dual bounding procedures used for pruning, but we also briefly discuss other aspects. Computational results for three example classes of PICP-W+ are given in Section 5.

2 Partial inverse combinatorial problems (PICPs)

2.1 Definition and notation

The partial inverse combinatorial problem (PICP) in the general form can be formulated as follows:

$$\begin{array}{lrcll}
\min\limits_{w,y} & \multicolumn{3}{l}{\sum\limits_{e\in E} |w_e|} & \\
\text{s.t.} & l^w & \le & w \quad \le u^w & \\
 & y_e & = & 1 & (\ e \in R\ ) \\
 & y & \in & \operatorname*{arg\,min}\limits_{y'\in Y} (d+w)^\top y', &
\end{array}$$

where the lower-level problem is a combinatorial problem whose feasible set $Y \subseteq \{0,1\}^n$ consists of 0-1 vectors and is independent of the upper-level variables. Upper-level variables w modify the lower-level objective function $(d+w)^\top y$, where the parameter vector d specifies initial weights and y are the lower-level characteristic vectors. By slight abuse of notation, we sometimes also identify y with the set $\{e \in E \mid y_e = 1\}$ if no confusion is possible. Note that the above problem formulation implies the optimistic assumption. On the upper level, upper-level variables w may be bounded from below by l^w and from above by u^w. If l^w and/or u^w are not specified, we assume that arbitrary weight modifications in the respective direction are allowed. We call R the *set of required elements*, i.e. the set of elements for which the leader has to ensure that they will be part of a lower-level optimal solution.

We use the ℓ_1-norm for measuring weight modifications, thus minimize $\sum_e |w_e|$ on the upper level. Depending on the application, a weighted sum might be more appropriate. We use unit weights for simplicity of notation, although a generalization of our algorithmic results to the weighted case is easily possible. This also holds if there are more general partial-solution constraints on the upper level that also exclude elements from a lower-level solution. If a specific lower-level solution is uniquely determined by R, the PICP is in fact a *complete* ICP.

We may assume that $l^w \leq 0 \leq u^w$ (possibly $+/-\infty$) by modifying d accordingly and adding a constant offset in the objective of the upper level. If $l^w = 0$, the problem only allows weights to be increased. We denote this by adding the suffix 'W+' to the name of the respective PICP class. Similarly, if $u^w = 0$, we have a PICP in which weights can only be decreased, denoted by the suffix 'W-'.

2.2 Optimality conditions

Next, we give some necessary conditions for an optimal solution of a PICP. They are a generalization of [29, Lemma 1].

Lemma 2.1. *Let (w^*, y^*) be an optimal solution. Then the following holds:*

1. *If $y^*_e = 1$ then $w^*_e \leq 0$.*
2. *If $y^*_e = 0$ then $w^*_e \geq 0$.*
3. $\sum_e |w^*_e| \geq \min_y \left\{d^\top y \mid y \in Y, y_e = 1 \;\forall e \in R\right\} - \min_y \left\{d^\top y \mid y \in Y\right\}$

In case the upper-level restrictions define a unique lower-level solution and the lower-level problem can be formulated as a linear program (LP), then also the inverse problem can be formulated as an LP. This result is based on duality theory [2].

Inequality 3 in the above lemma is satisfied with equality if the lower-level problem is an LP with all variables in the objective as can be shown based on duality [29, Theorem 3]. However, the inequality can be strict in general as the following example shows.

Example 2.2. Let $Y = \{\{A,B\},\{A,C\},\{B,D\},\{E\}\}$ with $d_A = d_B = 5, d_C = d_D = 2, d_E = 6$ and $R = \{A, B\}$. The only lower-level solution including R is the set $\{A, B\}$ with a total weight of 10. The minimal lower-level solution is $\{E\}$ with weight 6. However, there is no feasible weight modification of total weight $10 - 6 = 4$.

2.3 Easy lower-level problems

In the following, we consider PICPs with lower-level problems that can be solved as linear programming problems.

Assumption 2.3. We assume that the lower-level problem is given as a linear program such that for all objectives there is an optimal 0-1 solution.

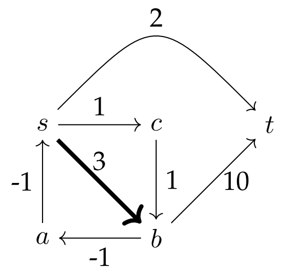

Figure 3: Reducing only the weight of the required arc sb in this instance of the PISPP will not give a bilevel-feasible solution due to the negative weights on arcs ba, as. The reduction of the weight of arc sb would need to be at least $2 - 10 - 3 = -11$ resulting in a negative cycle through s, b, a. An optimal solution is to reduce the weight of arc sb by one and decreasing the weight of arc bt by 10 to 0. Only reducing the weight of arc bt is also not a solution.

The corresponding ICPs where R defines a complete lower-level solution have been considered in [2]. Based on this assumption, we have the following problem formulation, where we use equality constraints and non-negative variables on the lower level.

$$\begin{aligned} \min_{w,y} \quad & \sum_{e\in E} |w_e| && \text{(PICP)} \\ \text{s.t.} \quad & l^w \leq w \leq u^w \\ & y_e = 1 && (\ e \in R\ ) \\ & y \in \operatorname*{arg\,min}_{y'} \left\{ (d+w)^\top y' : Ay' = b, y' \geq 0 \right\} \end{aligned}$$

For some lower-level problems, some (auxiliary) variables need to have an unmodifiable coefficient of zero in the lower-level objective. In order to include such lower-level problems using the above formulation, it can be necessary to prevent non-zero weight modifications for these variables by setting the according upper-level bounds to zero. This situation for example appears for the PIMCP, one of our three example classes of PICPs described below.

Example: assignment The assignment problem is to find a minimum-weight perfect matching in a bipartite graph. It can be formulated as the LP

$$\min_y \left\{ d^\top y : \sum_{y_e \in \delta(v)} y_e = 1\ \forall v \in V,\ y \geq 0 \right\}$$

which fulfils Assumption 2.3, e.g. [19, Theorem 11.4]. There are no variables that do not appear in the objective. The partial inverse assignment problem (PIAP) is to find minimal weight modifications such that there is a minimal assignment containing some specified edges.

Example: shortest path The shortest path problem is to find a minimal weighted path from a source s to a sink t. It can be formulated as an LP fulfilling Assumption 2.3 via one unit of flow leaving the source (and entering the sink) and flow conservation constraints at all non-terminal nodes. The partial inverse shortest path problem (PISPP) is to find minimal weight modifications such that there is a shortest s-t-path that contains some specified arcs.

This problem description can be problematic in case of cycles. Arcs that are required in PISPP may only be included within a 0-weight cycle as part of a solution in the above LP formulation. This does not solve PISPP as a path visits every node at most once. Furthermore, when regarding the shortest path problem only decreasing weights of required arcs can fail as the lower-level problem turns unbounded due to negative cycles, see Figure 3. Hence, for simplicity we add the following assumption.

Assumption 2.4. For a shortest path problem on the lower level, we assume that the considered graph is acyclic.

Example: min cut The minimum s, t-cut problem is to find a partition of the vertex set of a graph such that the total capacity of all arcs with tail in the set containing s and head in the set containing t is minimal.

Assumption 2.5. For a min cut problem on the lower level, we assume that all arc capacities are non-negative, i.e. $d + w \geq 0$ for all feasible w.

The following LP formulation of the min cut problem is based on having variables y_{uv} for all arcs and variables z_v for all vertices. It fulfils Assumption 2.3. A variable z_v indicates whether the vertex v is on the same side of the cut as the source s ($z_v = 1$) or the sink t ($z_v = 0$). For an arc (u, v), the variable z_{uv} indicates whether the arc is forward-crossing the cut and hence needs to be counted.

$$\begin{array}{llcll}
\min\limits_{y,z} & (d+w)^\top y & & & \\
\text{s.t.} & y_{uv} - z_u + z_v & \geq & 0 & (\ (u,v) \in E\ ) \\
 & y_{uv} & \geq & 0 & (\ (u,v) \in E\ ) \\
 & z_s & = & 1 & \\
 & z_t & = & 0 &
\end{array}$$

The dual of the min cut problem is the max flow problem. For a flow x_{uv}, flow conservation at all vertices except the source and sink is required. Furthermore, the flow on every arc is at most the arcs' modified capacity $d_{uv} + w_{uv}$.

The partial inverse min cut problem (PIMCP) is to find minimal modifications of capacities such that there is a minimal s, t-cut with some specified vertices on the same side as the source respectively the sink. This is the definition also used in [15] where this problem is shown to be $\mathcal{NP}$-hard.

An alternative definition would require some arcs to be (forward-)crossing the cut instead of fixing some vertices on either side of the cut. This implies that head and tail of these arcs have to be on the same side of the cut as the sink and source, respectively. Thus, the above definition fixing vertices is more general and includes the other one.

In contrast to the previous two example problems, the variables that are fixed on the upper level of the PIMCP are not those whose weight is modified. While some variables corresponding to vertices are fixed on the upper level, they are not part of the lower-level objective. The lower-level objective only depends on the variables corresponding to arcs. Weight modifications for the vertex variables have to be explicitly set to zero.

When formulating the inverse min cut problem as a single-level problem, we need to explicitly require that there is no flow on backward crossing arcs. Otherwise, there can be cycles of flow due to the requirement of saturated required arcs, see Figure 4. Forbidding flow on arcs of a partial cut that are already fixed as backward-crossing thus improves the formulation for the PIMCP.

$$\begin{tikzcd}
& & b \arrow[dr, "5"] & & \\
s \arrow[r, "1"] & a \arrow[ur, "5", line width=1pt] & & c \arrow[ll, "5"'] \arrow[r, "1"] & t
\end{tikzcd}$$

Figure 4: Partial Inverse Min Cut Problem (PIMCP) with arc ab required to cross the cut: Prevent that arcs sa, ab, bc, ct are saturated and ac has flow 4.

2.4 Existing literature

Related problems

A PICP is a specific type of bilevel problem with *coupling constraints*. These are constraints on the upper-level that explicitly depend on lower-level variables, see e.g. [18]. Alternative names for coupling constraints are *connecting constraints*, cf. [23], or *joint constraints*, cf. [7]. Coupling constraints pose several difficulties [23, 18]. For example, the inducible region of a bilevel continuous problem with coupling constraints is not necessarily connected. Many methods and implementations for bilevel linear problems do not allow the existence of coupling constraints, see [7]. The algorithm proposed by Fischetti et al. in [11] based on the high point relaxation can deal with coupling constraints. However, for this algorithm both lower-level objective and constraints have to be linear in both upper- and lower-level variables. In PICPs the lower-level objective is bilinear in upper- and lower-level variables.

In PICPs the lower-level objective depends on the upper-level variables, but the feasible area does not. Furthermore, the upper-level objective is independent of the lower-level variables. These are properties shared with a certain type of interdiction problems. Interdiction problems (or 'interdiction games') are bilevel problems in which the leader can forbid the follower to use certain structures such that the follower's optimal solution gets worse. There are two versions, with either the leader having a budget for interdiction or a target for the lower-level objective. Considering the budget version, there is no distinction between optimistic and pessimistic interdiction problems as both levels share the objective function. However, it is especially the target version that is structurally similar to PICPs. In contrast to PICPs, in interdiction problems the leader variables are typically binary. Allowing continuous interdiction seems to be more difficult [22]. Either the lower-level feasible set depends on the leader's decision which elements are interdicted, or a large extra-weight is added for the corresponding elements in the lower-level objective function [17]. In the latter case, the problem's structure is similar to the one of PICPs. Interdiction problems represent a particularly well-studied subclass of bilevel problems and several specialized methods have been developed for them. Interdiction problems can be solved via decomposition approaches that are strengthened by cuts based on the lower-level problem, e.g. [12, 14, 27]. Problems that have been considered as lower-level problem for interdiction include both problems solvable in polynomial time like shortest path [17] and matching [30], as well as $\mathcal{NP}$-complete problems like knapsack [12] or max clique [14]. For further references also see [18, 25, 27] and references therein.

(Complete) Inverse combinatorial problems (ICPs)

In the literature, the term *inverse combinatorial problem* (ICP) usually means that in fact a full follower solution is specified that shall become optimal by minimal weight modifications. In this paper, we often add the word *complete* for clarity and contrast to the *partial* in PICP.

Complete ICPs with a polynomial solvable problem on the lower level are solvable in polynomial time [2]. The proof in [2] can easily be adapted by changing the corresponding constraints to show the following theorem:

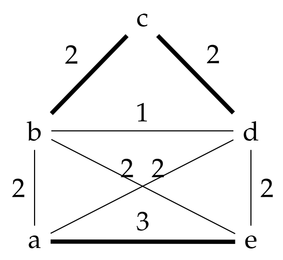

Figure 5: partial inverse min spanning tree problem with $R = \{ae, bc, cd\}$: Considering only the triangle bcd increasing the weight of edge bd by one is better than decreasing both edges bc, cd each by one. For the edge $ae \in R$ decreasing its weight by one is better than increasing either the two edges ab, ad or the two edges eb, ed each by one. In total, the minimal weight modification by either only increasing or decreasing is 3 while the optimum is increasing the weight of bd and decreasing the weight of ae has an objective value of 2.

Theorem 2.6. *If the lower level problem is solvable in polynomial time for every linear cost function, then the corresponding inverse problems with only weight increase (ICP-W+) respectively only weight decrease (ICP-W-) are also solvable in polynomial time.*

In case of an LP on the lower level, a single-level reformulation based on strong duality or KKT conditions simplifies to an LP for a complete lower-level solution. Combinatorial methods for ICPs are often based on problem-specific solution methods for the underlying combinatorial problem or closely related problems. For example, the inverse shortest path problem can be reduced to a shortest path problem and the inverse assignment problem can be solved via an assignment problem [2]. The inverse min cut problem can be solved based on a max-flow evaluation [3]. Algorithms for some ICPs with additional integer requirements are given in [1]. For further results on ICPs, the reader is referred to the surveys [16, 8] and references therein.

Complete ICPs with a problem that is $\mathcal{NP}$-complete on the lower level are co$\mathcal{NP}$-complete [5]. There are solution methods based on enumeration of the lower-level feasible solutions [26, 4].

Partial inverse combinatorial problems (PICPs)

In contrast to the (complete) ICP, the PICP is $\mathcal{NP}$-hard for a general problem that is solvable in polynomial time. Table 1 provides an overview of complexity results for some particularly well-known combinatorial problems of that type. Furthermore, even if a version of a PICP happens to be solvable in polynomial time, it may become $\mathcal{NP}$-hard once only weight increases are allowed or more general bounds on the weight modifications are added. To the best of our knowledge, no solution methods tailored to PICPs that are $\mathcal{NP}$-complete have been proposed in the literature.

Checking feasibility of a given solution in case the lower-level problem is solvable as an LP can be done by simply adding constraints fixing the required parts of the solution to the LP and comparing to an optimal lower-level solution. Combinatorial methods for some problems based on considering an accordingly smaller graph are given in Section 4.1 below.

When discussing PICPs, one has to distinguish whether weights can be changed arbitrarily or whether they are to be only increased or only decreased. In general, none of these cases is reducible to another. For an example of the partial inverse min spanning tree problem where neither exclusively decreasing nor increasing can produce an optimal solution, see Figure 5. However, if there are no constraints on the weight modifications, for several underlying problems – like shortest path (without negative cycles), assignment or min cut – it suffices to only decrease weights [2]. If the underlying problem allows assuming without

	W-		W+	
Min Spanning Tree	$\mathcal{P}$	[6]	$\mathcal{NP}$-complete	[21]
Min Spanning Tree with $\lvert R\rvert = 1$	$\mathcal{P}$		$\mathcal{P}$	[21]
Min Basis of Matroid	$\mathcal{P}$	[31]	$\mathcal{NP}$-complete	
Shortest path (already feasibility)	$\mathcal{NP}$-complete	Theorem 2.8	$\mathcal{NP}$-complete	
Shortest path on DAG	$\mathcal{P}$	Theorem 2.9	$\mathcal{NP}$-complete	
Shortest path on DAG with $\lvert R\rvert = 1$	$\mathcal{P}$		$\mathcal{NP}$-complete	Theorem 3.1
LP in standard form with opt. 0-1 solution	$\mathcal{P}$	[29]	$\mathcal{NP}$-complete	
Assignment	$\mathcal{P}$	[29]	$\mathcal{NP}$-complete	
Assignment with $\lvert R\rvert = 1$	$\mathcal{P}$		$\mathcal{NP}$-complete	Theorem 3.2
Min cut	$\mathcal{NP}$-complete	[15]	$\mathcal{NP}$-complete	

Table 1: Complexity results for different PICPs. For problems below the middle horizontal line, results for W- also apply to general weight modifications.

loss of generality that weights will only be decreased in the (complete) ICP, then this remains true for the corresponding PICP. This result has been established for the PIMCP based on the respective result for the complete inverse min cut problem [15, Lemma 3.1]. Similarly, it can directly be adapted for assignment and shortest path problems based on the corresponding complete ICPs [2].

In general, only changing the weights of required elements in R may not yield an optimal solution of a PICP as the following example shows. It is also easy to see that only increasing weights may not yield an optimal solution.

Example 2.7. Let the lower-level feasible solutions be the three sets $\{A, B\}, \{A, C\}, \{B, C\}$ with original weights $d_A = d_B = 2, d_C = 1$, and $R = \{A, B\}$. This is already a complete solution. When only adapting the weights of the required elements, we need to reduce the weight of both A and B by one each, i.e. have an objective value of two. However, only increasing the weight of the third element C by one is also feasible and has an objective value of one.

Partial inverse combinatorial problems with only weight decreases (PICP-W-)

Polynomial algorithms are known for some PICPs where weights can only be decreased (PICP-W-). They are based on first evaluating a complete lower-level solution and then modifying the weights accordingly. In particular, a minimal lower-level solution fulfilling the partial-solution constraints on the upper level is used in these methods.

This approach works for the minimum spanning tree problem [6] and more generally for the minimum basis problem in a matroid on the lower level [31]. An optimal lower-level completion can be found by a greedy approach. Weight modifications are evaluated using the fundamental circuit or fundamental cocircuit of required elements.

Furthermore, such a minimal completion can be used if the lower-level problem can be formulated as a linear program in standard form $\min_x \left\{c^\top x \mid Ax = b, x \geq 0\right\}$ such that for all weight functions c an optimal 0-1 solution exists [29]. Based on complementary slackness and duality of LPs, first R is completed to a feasible lower-level solution. Then the according weight modifications are evaluated. For example, PIAP can be solved with this method [29].

This LP-based method might include unnecessary lower-level substructures that get a modified weight of 0. For example, cycles of zero weight can be included in the lower-level solution when applying this method to PISPP or PISPP-W-. This is not feasible in the combinatorial sense. The inclusion of cycles can be prevented by adding constraints to the LP that at most one outgoing arc per vertex is chosen. However, then the LP in standard form includes slack variables that are not part of the objective function. In contrast, the LP-based method mentioned above assumes that all lower-level variables are included with weights that can be modified in the objective function [29].

It is already $\mathcal{NP}$-hard to determine feasibility of an instance of PISPP on a general graph. A cycle-free solution only exists if there are vertex-disjoint paths connecting the respective parts with source and sink. $\mathcal{NP}$-completeness for PISPP thus follows from the $\mathcal{NP}$-completeness of the vertex-disjoint path problem, e.g. [6].

Theorem 2.8. *PISPP on a directed graph is $\mathcal{NP}$-complete.*

Proof. An instance of PISPP on a directed graph is only feasible if there are vertex-disjoint paths connecting the required arcs with the source and sink. For $R = \{ab\}$ with $s \neq a$ and $b \neq t$, we need vertex-disjoint s-a- and b-t-paths. This is the 2-paths-problem which is $\mathcal{NP}$-complete [13, Lemma 3]. □

In contrast, the existence of the corresponding vertex-disjoint paths can easily be determined on a directed acyclic graph. Moreover, optimal weight reductions for PISPP on a DAG can be evaluated in polynomial time if weights of arcs in R can be reduced arbitrarily.

Theorem 2.9. *PISPP on a DAG is solvable in polynomial time by only reducing the weights of required arcs.*

Proof. If $|R| = 1$, one first evaluates a shortest path and a shortest path including R. Reducing the weight of the required arc by the weight difference of the shortest path completion and the shortest path solves the problem. If $|R| > 1$, PISPP can be solved by iteratively reducing the weights of the required arcs using a topological ordering. Starting with the arc $a_1 \in R$ that is closest to the source, use the tail of the second closest required arc $a_2 \in R$ as preliminary sink to reduce the weight of a_1. Continue by reducing the weight of a_i by using the tail of the next-closest arc a_{i+1} as preliminary sink until the sink is reached. Observe that a first part of a shortest path using a required arc is given by the previous iteration's path. Thus, PISPP on a DAG is solvable via a number of shortest path evaluations linear in $|R|$ if arbitrary weight decreases are allowed. □

Similarly to PISPP on general graphs, PIMCP is $\mathcal{NP}$-complete and cannot be solved via the LP-based method. The min cut problem can be formulated as an LP with variables corresponding to the vertices and to the arcs. Only variables corresponding to arcs are included with nonzero weight in the objective function and only these weights can be modified. The min cut problem is a problem where some weights are fixed to zero and cannot be modified. Thus, the LP-based method described above cannot be applied. PIMCP is $\mathcal{NP}$-complete also when weight decreases for the arcs are allowed [15].

Partial inverse combinatorial problems with only weight increases (PICP-W+)

The only PICP with only weight increases (PICP-W+) that has been studied so far is, to the best of our knowledge, the partial inverse min spanning tree problem. If $|R| = 1$, it is solvable in polynomial time [21]. As soon as $|R| > 1$, the partial inverse min spanning tree problem with only weight increases is $\mathcal{NP}$-complete [21]. We show in Section 3 that for both the shortest path problem on a directed acyclic graph and the assignment problem, the corresponding PICP-W+ is already $\mathcal{NP}$-complete if $|R| = 1$.

Other norms

Instead of minimizing the ℓ_1-norm of the modifications, other objective functions are possible, e.g. see [16] and references therein. PICPs with ℓ_∞ instead of ℓ_1 are considered as preprocessing problems for underlying problems like the shortest path problem or the assignment problem in [20].

2.5 Solution approaches

Bilevel problems like PICPs can be solved via singe-level reformulations, decomposition approaches or branch-and-bound schemes. We shortly explain single-level-reformulation and decomposition approaches for PICPs in the following. Details on our branch-and-bound scheme are given in Section 4.

Single-level reformulation: via KKT conditions or strong duality

A single-level reformulation of a PICP where the lower-level problem can be written as a parametric LP can be obtained by replacing the lower-level problem by its primal and dual constraints as well as optimality conditions. For the latter, one can use the KKT conditions/complementarity or strong duality. In the first case, there are many rather simple, local bilinear constraints while in the latter case only one, however, more complicate bilinear constraint is added. The bilinearity of the strong duality constraint stems from the bilinearity of the lower-level objective function when simultaneously regarding both upper- and lower-level variables.

In the dual of the lower-level problem, the feasible region is dependent on the upper-level variables w, instead of the objective. The dual lower-level problem with dual variables x is the following LP:

$$\begin{array}{lll} \max\limits_{\lambda} & \lambda^\top b & \\ \text{s.t.} & \lambda^\top A & \leq \quad d + w. \end{array}$$

While the primal lower-level objective is bilinear in the upper- and lower-level variables, the dual lower-level constraints are linear in both the upper- and lower-level variables.

Optimality conditions based on complementary slackness are

$$0 = y_e(\lambda^\top A_e - d_e - w_e) \quad (e \in E),$$

where e are the indices of the primal variables. Combining the lower-level primal and dual constraints with the optimality conditions, we obtain the following single-level reformulation for the PICP:

$$\begin{array}{lrcll} \min\limits_{w,y,\lambda} & \sum\limits_{e\in E} |w_e| & & & \\ \text{s.t.} & y_e & = & 1 & (\ e \in R\ ) \\ & l^w & \leq & w \quad \leq u^w & \\ & \lambda^\top A & \leq & d + w & \\ & Ay & = & b & \\ & 0 & = & y_e(\lambda^\top A_e - d_e - w_e) & (\ e \in E\ ) \end{array} \tag{KKT-1level}$$

Instead of using complementary slackness as optimality condition, one can also require strong duality to hold, i.e. require that primal and dual objective have the same value by adding the constraint

$$\lambda^\top b = (d + w)^\top y.$$

If a complete lower-level solution y is fixed on the upper level, the nonlinear constraints in the above single-level reformulation simplify to linear (in some cases trivial) constraints. The two approaches via complementary slackness or strong duality then both result in an LP.

In case of only a partial lower-level solution being required on the upper level, bilinear constraints remain. In the reformulation via complementary slackness, there is one bilinear constraint per lower-level variable that is not fixed on the upper level. When we only keep those bilinear constraints that simplify to linear constraints by plugging in the lower-level variables that are fixed on the upper level and remove all other bilinear constraints, we obtain an LP relaxation of the original problem. We will later use this relaxation to obtain lower bounds, see Section 4.3.2.

The nonlinear single-level reformulation can be formulated as a mixed-integer linear program via big-M constraints. However, weight modifications do not necessarily have upper bounds in the problem formulation. When implementing this approach for comparison, we therefore use indicator constraints for the solver.

Decomposition

PICP can be reformulated by enumerating the finite number of lower-level feasible solutions. This decomposition approach is a generalization of a cutting plane algorithm for inverse mixed-integer linear programs [26, 4]. It is similar to cutting plane algorithms for interdiction problems, e.g. [9].

For every lower-level feasible solution, the lower-level objective has to be at least as large as for a lower-level feasible solution that also fulfils the partial-solution constraints on the upper level.

$$\begin{array}{lrcll} \min\limits_{w,y} & \sum\limits_{e\in E} |w_e| & & & \\ \text{s.t.} & y_e & = & 1 & (\ e \in R\ ) \\ & l^w & \leq & w \quad \leq u^w & \\ & Ay & = & b & \\ & (d+w)^\top y & \leq & (d+w)^\top z & (\ z \in \{z \in \{0,1\}^n : Az = b\}\ ) \end{array}$$

As the lower-level objective is bilinear, we get a program with bilinear constraints. These constraints can be reformulated using big M-constraints. For implementing these constraints, one can use indicator constraints provided by the used solver.

Instead of enumerating all constraints for lower-level feasible solutions z, it suffices to only add those that are otherwise violated. This can be done by a decomposition approach. In order to find constraints that have to be added to the master problem, solve the lower-level problem with the current best weight modifications. If the obtained lower-level solution has an objective that is less than the one of the current lower-level solution fulfilling the partial-solution constraints, the corresponding constraint has to be added to the master problem. In case the objectives are the same, the optimal weight modifications have been found and we terminate.

Using combinatorial algorithms to solve the lower-level problem can be advantageous, especially if several lower-level optimal solutions are evaluated. For example, for the shortest path problem, it is possible to obtain all shortest paths by using Dijkstra's algorithm with only small extra effort in comparison to evaluating a single shortest path. Evaluating a max flow for the min cut problem allows finding two different minimal cuts at once. Adding all corresponding constraints at once to the master problem results in a better performance.

2.6 Pessimistic PICPs

So far and after this subsection, the optimistic version is considered. For PICPs, solving the optimistic or the pessimistic version is not much different from an algorithmic point of view though there are different objectives on the two levels.

Theorem 2.10 (Pessimistic PICP). *Let (w^*, y^*) be an optimal solution to an optimistic PICP, PICP-W+ or PICP-W-. Then we have:*

1. *If $w^* \neq 0$, a pessimistic optimum is not attained.*
2. *If no proper subset of y^* is again a feasible lower-lever solution, then for every $\varepsilon > 0$ there is a pessimistic feasible solution with value $|w^*| + \varepsilon$ that is easily computable from (w^*, y^*).*

Proof. 1. Let $w^* \neq 0$ and assume that $(\hat{w}, \hat{y})$ is a pessimistic optimal solution. This implies that we have $(d + \hat{w})^\top \hat{y} < (d + \hat{w})^\top y$ for all $y \in Y$. Since the inequality is strict, it still holds for all w with $\|\hat{w} - w\| \leq \varepsilon$ for ε sufficiently small. Due to $w^* \neq 0$, also $\hat{w} \neq 0$ since a pessimistic optimum cannot have a better objective value than an optimistic optimum. We can therefore choose $\bar{w}$ between 0 and w with $\|\hat{w} - \bar{w}\| \leq \varepsilon$. Therefore, $(\bar{w}, \hat{y})$ is still feasible in all three problem cases PICP, PICP-W+ and PICP-W-, and also strictly improves the upper-level objective, contradicting that $(\hat{w}, \hat{y})$ was assumed to be a pessimistic optimal solution.

2. Since (w^*, y^*) is an optimistic optimal solution, we have $(d + w^*)^\top y^* \leq (d + w^*)^\top y$ for $y \in Y$. For PICP and PICP-W+, we distribute ε on all elements that are not used in the optimal solution y^*, i.e. for $m = |\{e \mid y^*_e = 0\}|$, set $w^p_e = w^*_e + \varepsilon/m$ if $y^*_e = 0$ and $w^p_e = w^*_e$ otherwise. The lower-level objective value of y^* does not change but is now by at least ε smaller than the objective of any other $y \in Y$, as every other feasible lower-level solution contains at least one element that is not part of y^*. Thus, (w^p, y^*) is a pessimistic bilevel-feasible solution with upper-level objective value $|w^p| = |w^*| + \varepsilon$. For PICP-W- (and PICP), a similar construction can be applied, decreasing the weight of all elements of y^* that are also in R by an equal share of ε.

□

The condition in the first part of Theorem 2.10 is easy to check for. This is also true for the condition in the second part once y^* is available. The latter condition is necessary since in the opposite case, i.e. if there is a lower-level solution $\bar{y} \in Y$ that is strictly contained in y^*, every additional weight put onto $\bar{y}$ also affects y^*, so y^* cannot be turned into a pessimistic optimal solution. In fact, the pessimistic problem can be infeasible if only y^* but not $\bar{y}$ is bilevel-feasible. As can be seen from the proof of Theorem 2.10, the assumption is only necessary for PICP-W+ since PICP and PICP-W- offer the option of decreasing weights for elements in y^* that are not in $\bar{y}$.

3 Complexity results

3.1 PISPP with only weight increases (PIAP-W+) on a DAG

Theorem 3.1. *The partial inverse shortest path problem (PISPP) on a directed acyclic graph with only weight increases is $\mathcal{NP}$-complete, even if $|R| = 1$.*

The $\mathcal{NP}$-completeness of PISPP-W+ with $|R| = 1$ is in contrast to the analogue case in the partial inverse min spanning tree problem, which is solvable in polynomial time [21]. The following proof is an adaption of the proof for $\mathcal{NP}$-completeness of the preprocessing shortest path problem on a DAG [20]. In comparison to PISPP, the preprocessing shortest path problem is to minimize the maximal modification of any arc instead of the sum of the absolute changes.

Proof. The proof is a reduction from the well-known $\mathcal{NP}$-complete problem 3-SAT. Let an instance of 3-SAT is given by its variables $X = \{x_1, ..., x_n\}$ and clauses $B = \{B_1, ..., B_m\}$ with each clause consisting of (up to) three literals $B_j = \{b_{j1}, b_{j2}, b_{j3}\}$ for $j \in [m]$. In the following, we create an acyclic directed graph $G = (V, A)$ with costs d such that satisfying assignments for the 3-SAT instance correspond to s_0-t_m-paths that use arc (s_n, t_0), and vice versa. In the adaption for PISPP, other weights for the so-called shortcut arcs are used in comparison to the proof in [20]. Furthermore, we simplified the graph by not including arcs between the variable- and clause-paths and renamed the auxiliary vertices. Direct conclusions of the construction that do not depend on the weights given in [20] remain true.

Construction of instance We construct a graph consisting of two parts, the first corresponding to the variables and the second part corresponding to the clauses. These two parts are similarly structured parallel paths via $x_i, \overline{x_i}$ and b_{j1}, b_{j2}, b_{j3} respectively. We denote both the literal and its corresponding vertex in the constructed instance of PISPP-W+ with b_{jk} as it should be clear which is meant. The parts for the respective variables and clauses are connected with auxiliary variables $s_0, ..., s_n$ and $t_0, ..., t_m$ respectively. The two parts are connected by one arc (s_n, t_0) that is the unique element in R. For a sketch, see Figure 6.

The first part of the graph includes vertices $x_i, \overline{x_i}$ and $s_0, ..., s_n$ connected by $4n$ arcs (s_{i-1}, x_i), (x_i, s_i), $(s_{i-1}, \overline{x_i})$, $(\overline{x_i}, s_i)$ for $i \in [n]$. An s_0-s_n-path corresponds to a truth assignment as every such path either includes x_i or $\overline{x_i}$ for all $i \in [n]$.

The second part of the graph corresponding to the clauses consists of vertices $t_0, ..., t_m$ between 'parallel' vertices $b_{jk}, j \in [m], k \in [3]$. They are connected by arcs $(t_{j-1}, b_{jk}), (b_{jk}, t_j)$ for $j \in [m], k \in [3]$ such that for every clause $B_j = \{b_{j1}, b_{j2}, b_{j3}\}$ there are three parallel paths from t_{j-1} to t_j via the three literals respectively. A t_0-t_m-path corresponds to choosing one literal from every clause. So far, all arcs have unit weights d and costs c are one unit per unit weight increase.

Furthermore, for every literal b_{jk} a shortcut arc is added from x_i if $b_{jk} = \overline{x_i}$ and from $\overline{x_i}$ if $b_{jk} = x_i$. Shortcut arcs (x_i, b_{jk}) or $(\overline{x_i}, b_{jk})$ obtain a weight of $2(n-i+j)$. Then the corresponding path using a shortcut arc is exactly one unit shorter than the path via (s_n, t_0). Assume that the cost c for weight increase of a shortcut arc is $n + 2m + 1$. Replace shortcut arcs by the according number of parallel arcs and introduce auxiliary 'inbetween'-vertices.

In total, we have:

$$
\begin{aligned}
V(G) &= \{x_i, \overline{x_i} : i \in [n]\} \cup \{s_0, ..., s_n\} \cup \{b_{j1}, b_{j2}, b_{j3} : j \in [m]\} \cup \{t_0, ..., t_m\} \\
A(G) &= \{(s_{i-1}, x_i), (x_i, s_i), (s_{i-1}, \overline{x_i}), (\overline{x_i}, s_i) : i \in [n]\} \\
&\quad \cup \{(t_{j-1}, b_{jk}), (b_{jk}, t_j) : k \in [3], j \in [m]\} \\
&\quad \cup \{(s_n, t_0)\} \\
&\quad \cup \{(x_i, b_{jk}) : \overline{x_i} = b_{jk}, i \in [n], j \in [m], k \in [3]\} \\
&\quad \cup \{(\overline{x_i}, b_{jk}) : x_i = b_{jk}, i \in [n], j \in [m], k \in [3]\} \\
R &= \{(s_n, t_0)\} \\
d(a) &= \begin{cases} 2(n-i+j), & \text{if } a \in \{(x_i, b_{jk}), (\overline{x_i}, b_{jk})\} \\ 1, & \text{else} \end{cases} \\
c(a) &= \begin{cases} n + 2m + 1, & \text{if } a \in \{(x_i, b_{jk}), (\overline{x_i}, b_{jk})\} \\ 1, & \text{else} \end{cases}
\end{aligned}
$$

The construction can be done in polynomial time.

By construction, the following observations hold:

1. Vertices $x_i, \overline{x_i}$ have exactly one ingoing arc $(s_{i-1}, x_i), (s_{i-1}, \overline{x_i})$ respectively.
   Vertices b_{jk} have exactly one outgoing arc (b_{jk}, t_j).

2. Every directed path P in G from node s_0 to t_n includes either arc (s_n, t_0) or includes exactly one shortcut arc.

3. Every s_0-t_m-path P using arc (s_n, t_0) has (original) weight $d(P) = 2n + 2m + 1$.
   Every s_0-t_m-path Q using a shortcut arc has (original) weight $d(Q) = 2n + 2m$.

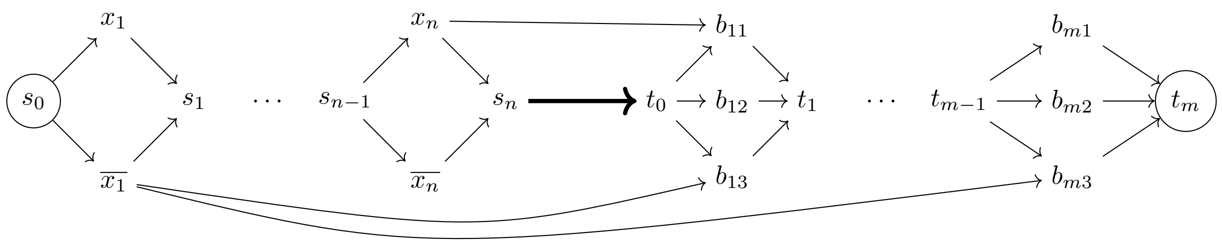

Figure 6: Sketch of complexity proof graph with $x_1 = b_{13}$, $x_1 = b_{m3}$, $\overline{x_n} = b_{11}$

4. Let P be an s_0-t_n-path containing arc (s_n, t_0). Then $|P \cap \{x_i, \overline{x_i}\}| = 1$ for every $i \in [n]$ and for every $j \in [m]$ there is a $k \in [3]$ such that $b_{jk} \in P$.

Claim The 3-SAT instance is satisfiable if and only if there is an s_0-t_n-path P in G and weight modifications w that fulfil the following three conditions:

1. The arc (s_n, t_0) is part of P,
2. the weighted additional costs are at most $n + 2m$, so $\sum_{a \in A} c_a w_a \leq n + 2m$, and
3. P is a shortest path in G with weights $d + w$.

Forward-direction Let x^* be a satisfying truth assignment (more than one such x^* might exist). At least one literal of every clause is satisfied as x^* is a satisfying truth assignment by assumption. In order to simplify notation, reorder literals of clauses such that b_{j1} is satisfied by x^* for every $j \in [m]$.

Let V^* bet the following (sub)set of true literals

$$V^* = \{x_i : x_i^* = 1\} \cup \{\overline{x_i} : x_i^* = 0\} \cup \{b_{j1} : j \in [m]\}.$$

Then there is a unique s_0-t_n-path P through all vertices in V^* (and all vertices $s_i, t_j, i \in [n]_0, j \in [m]_0$). Set additional weights w to one for edges (s_i, x_i) and $(s_i, \overline{x_i})$ that are not part of P as well as for all arcs $(b_{j2}, t_j), (b_{j3}, t_j), j \in [m]$. Let $w(a) = 0$ for all remaining arcs, in particular all shortcut arcs:

$$w(a) = \begin{cases} 1, & a \in \{(s_i, x_i) : x_i \notin V^*\} \cup \{(s_i, \overline{x_i}) : \overline{x_i} \notin V^*\} \cup \{(b_{jk}, t_j) : j \in [m], k \in \{2, 3\}\} \\ 0, & \text{else.} \end{cases}$$

First, we argue that (s_n, t_0) is part of P. By construction P is via b_{11} and either x_n or $\overline{x_n}$. Thus, P could only include a shortcut arc (x_n, b_{11}) or $(\overline{x_n}, b_{11})$ preventing that (s_n, t_0) is part of P. If $x_n \in V^*$, the existence of a shortcut arc (x_n, b_{11}) would imply that $b_{11} = \overline{x_n}$, contradicting the construction of V^*. Analogously, there is no arc $(\overline{x_n}, b_{11})$ in case $\overline{x_n} \in V^*$. The only path from x_n or $\overline{x_n}$ to b_{11} without a shortcut arc is via (s_n, t_0).

Second, we need that the total weighted additional costs are at most $n + 2m$. Corresponding to the n variables, we either have $w(s_i, x_i) = 1$ or $w(s_i, \overline{x_i}) = 1$ and for every of the m clauses there are exactly two arcs (b_{jk}, t_j) with additional weight of one each. For all remaining arcs, there is no weight modification. The total additional weight is $n + 2m$.

As third property, we need to prove that P is a shortest path in G with respect to weights $d + w$. By construction there is no additional weight for all arcs of P. Thus, the weight of P with respect to $d + w$ is the same as with respect to d. Additionally using the third observation, we obtain

$$(d + w)(P) = d(P) = 2n + 2m + 1.$$

In the following we argue that any other s_0-t_m-path Q has at least the weight of P. If Q contains no shortcut arc, then the modified weight of Q is at least as large as the one of P as weight modifications are non-negative.

For the remaining, assume Q contains a shortcut arc. Then the original length of Q is one unit shorter than P, so $d(Q) = 2n + 2m$. According to the second observation, exactly one shortcut arc is part of Q. There are six possible forms for a shortcut arc: $(\overline{x_i}, b_{jk})$ or (x_i, b_{jk}) for $k \in \{1, 2, 3\}$. We argue that in every case, there exists at least one arc a along Q with $w(a) = 1$, such that Q is at least as long as P.

First, we consider a shortcut arc (x_i, b_{j1}). By construction, we have $b_{j1} = \overline{x_i}$. According to the first observation, (s_i, x_i) is the only ingoing arc of x_i, so $(s_i, x_i) \in Q$. Due to the reordering of clause-literals, we know that b_{j1} is fulfilled, $x_i^* = 0$. The additional weight for arc (s_i, x_i) is thus set to one, $w(s_i, x_i) = 1$, such that Q has at least same length as P.

For a shortcut arc $(\overline{x_i}, b_{j1})$ being part of Q the argumentation is similar as in the previous case. By construction, we have $b_{j1} = x_i$ and $x_i^* = 1$. Again, according to the first observation, the arc $(s_i, \overline{x_i})$ is part of Q as it is the only ingoing arc of $\overline{x_i}$. We have set $w(s_i, \overline{x_i}) = 1$, so Q is at least as long as P.

In the final case, let the shortcut arc be $(\overline{x_i}, b_{jk})$ or (x_i, b_{jk}) for $k \in \{2, 3\}$. By construction, as noted in the first observation, the only outgoing arc of b_{jk} is (b_{jk}, t_j) which is thus part of Q. The additional weight of this arc (b_{jk}, t_j) is set to one, such that Q has at least the length of P.

Every satisfying truth assignment x^* thus corresponds to a solution of PISPP-W+ in the described instance.

Backward direction Let w_a be arc weights with $\sum_a c_a w_a \leq n+2m$ such that P is a shortest s_0-t_m-path with respect to $d + w$ and $(s_n, t_0) \in P$. By the second observation, P then contains no shortcut arc. Additionally, using the fourth observation, we have that P contains x_i or $\overline{x_i}$ for all $i \in [n]$. Define a truth assignment x^* by $x_i^* = 1$ if and only if $x_i \in P$.

We claim that x^* satisfies every clause B_j. We can assume $w(a) = 0$ for all arcs a of P as $w \geq 0$. Furthermore, there are also no weight increases for shortcut arcs, due to costs of $n + 2m + 1$ per unit weight increase on shortcut arcs. Exactly one $b_{jk}, k \in [3]$ is part of P for every $j \in [m]$ by the fourth observation.

Let b_{jk} be part of path P, then b_{jk} is fulfilled by the above-defined truth assignment. First, assume $b_{jk} = \overline{x_i}$. Then by construction (x_i, b_{jk}) is a shortcut arc. If $x_i \in P$, then using the shortcut arc would reduce the length of the path, which is a contradiction to P being a shortest path. Thus $\overline{x_i} \in P$ and $x_i \notin P$ and we have set $x_i^* = 0$ such that b_{jk} is fulfilled. Similarly, for $b_{jk} = x_i$, we get that b_{jk} is fulfilled as $x_i \in P$ so $x_i^* = 1$.

Thus, a solution of PISPP-W+ satisfying the listed conditions corresponds to a satisfying truth assignment x^*. □

3.2 PIAP with only weight increases (PIAP-W+)

Theorem 3.2. *The Partial Inverse Assignment Problem with only weight increases (PIAP-W+) is $\mathcal{NP}$-complete even if $|R| = 1$.*

The PIAP with arbitrary weight modifications is solvable in polynomial time, similar to PISPP on a DAG [29]. The solution is based on only reducing weights of arcs in $|R|$. Our result for PIAP-W+ is a strengthening of the $\mathcal{NP}$-completeness-result for PIAP with general bounds on the allowed weight modifications [28]. The reduction from PISPP-W+ to PIAP-W+ is based on the analogue problems under ℓ_∞ instead of ℓ_1 [20, Theorem 7].

Proof. First, feasibility of a PIAP-W+ solution can be determined in polynomial time. This requires solving two assignment problems on the graph with the according weight modifications – one of them fixing R to be part of the solution – and comparing their objective value.

We prove $\mathcal{NP}$-hardness of PIAP-W+ with $|R| = 1$ by a reduction from PISPP-W+ on a DAG with $|R| = 1$, which we have just shown to be $\mathcal{NP}$-hard, see Theorem 3.1. Let an instance for PISPP-W+ on a weighted acyclic digraph with two distinct vertices s, t and $R = \{a\}$ consisting of a single required arc be given. Without loss of generality we assume that the source s has only outgoing arcs and the sink t has only ingoing arcs. We split all other nodes into two nodes with all ingoing respectively outgoing arcs incident and connect them by a new *node splitting edge* each. Let A_{NS} be the set of all such node-splitting edges. We set the weight for node-splitting edges to 0, while all remaining edge weights remain unchanged. For PIAP-W+ we will ignore all arc directions. By construction the obtained undirected graph is bipartite.

If the terminal nodes s, t were removed, choosing all node-splitting arcs would constitute the minimal assignment (of weight 0). However, if s, t are included, the optimal solution is given by all non-node-splitting arcs of a shortest s-t-path and the node-splitting arcs for the remaining nodes, which are not on the s-t-path. This assignment is obtained as the symmetric difference of the path (considered as a set of arcs) with the set A_{NS}.

Without loss of generality, we may assume that only weights of arcs not in A_{NS} will be increased in an optimal solution of PIAP-W+. One could assure that by replacing every edge in A_{NS} by sufficiently many parallel edges, e.g. as many as the degree of the corresponding node.

One can now easily show that from an s-t-path we can easily construct an assignment with the same weight and vice versa: Given an s-t-path, the symmetric difference with the node splitting arcs results in the none-node splitting arcs of the path and node-splitting arcs for all nodes that are not on the path. Due to arcs in A_{NS} having weight 0, the assignment has the same weight as the path.

Given an assignment, the symmetric difference with A_{NS} results in an s-t-path after adding node-splitting arcs for vertices that are incident to non-node-splitting arcs in the assignment. Note that the orientation of arcs in the path are correct as by construction for each node exactly one ingoing and one outgoing arc must be part of the assignment. Furthermore, no cycles can be present in addition to the path as the original graph was assumed to be acyclic. By an analogue argumentation as above, the assignment and path have the same weight. Let a in the PISPP-W+ instance be also be required in the PIAP-W+ instance. Due to the direct correspondence of solutions described above, we conclude that PIAP-W+ is $\mathcal{NP}$-complete. □

4 Branch and bound for PICP-W+

We propose to solve PICP-W+ by a branch-and-bound method. We branch on the (upper-level-copies of the) lower-level variables, extending the set of required elements R for branch-and-bound nodes lower in the tree. Each node n in the branch-and-bound tree corresponds to a PICP-W+ for some $R^{(n)} \supseteq R$, while each leaf is a complete ICP. An optimal solution (w^*, y^*) will therefore be contained in the leaf representing y^* and each subtree for which the corresponding $R^{(n)}$ is contained in y^*. As usual for branch-and-bound methods, we hope to obtain strong bounds that will allow us to prune parts of the tree at an early stage. Based on problem knowledge, in every branching step several lower-level variables may be fixed at once. Adding more coupling constraints to the problem formulation might sound counter-intuitive since some solution methods for bilevel programs rather have to assume that there are no coupling constraints. However, here the added coupling constraints fix variables such that the number of free variables is reduced. Furthermore, the used methods for dual bounds turn out to be powerful, also because some information can be re-used. At the leaves of the branch-and-bound tree, where a complete lower-level solution is fixed, the single-level reformulation simplifies to a linear program.

The main focus of our algorithm lies on the novel bounding procedures. At any node, we may solve a subproblem for obtaining a primal bound. This bound turns out to be quite strong even for the root node, so we have a good incumbent right from the start, see Section 4.2 below. In fact, this heuristic picks a specific leaf of the branch-and-bound tree and evaluates it beforehand. Dual bounding strategies will be described in Section 4.3, where one type of subproblem will provide reasonable bounds for low-depth nodes while the other becomes tight when approaching leaves. Bounding procedures are applicable to general PICP-W+ of type (PICP) and form the basis for a general branch-and-bound method. However, we also briefly discuss suitable branching, node selection and propagation strategies that we use for our prototype implementation. Problem-specific components and intuitions are explained on the examples of the partial inverse shortest path, assignment and min cut problems. For a summary of the proposed method, see Algorithm 2.

Example: shortest path In case of PISPP-W+, the lower-level variables indicate for every arc of the graph whether it is part of the considered path. At a branch-and-bound tree node, one or several subpaths are required.

Instead of branching on whether a single arc is part of a path, we branch on sets from which exactly one arc is part of a path. Any inclusion-wise minimal s, t-cut provides a set of arcs of which exactly one is part of any lower-level solution. Note that not every arc in such an s, t-cut can be part of a path as there might be no connection to other required parts. We can reduce the number of arcs for which no connection to other required parts exists by using the in- respectively outgoing arcs of a vertex at either end of a so far required subpath, the source or the sink. Still, there can be arcs such that no completion to a feasible path exists such that the corresponding subtrees can be pruned once infeasibility is noticed.

When branching at the in-/outgoing arcs, there is one child for every arc in the branch-and-bound tree. The variable corresponding to the chosen arc is set to 1 and the variables corresponding to the other incident arcs is set to 0. Variables corresponding to arcs that are not part of any feasible path can be fixed as well. Notice that the obtained branch-and-bound tree is in general not binary. The number of children is the in-/out-degree of the vertex where branching takes place. The depth of the branch-and-bound tree is the number of arcs on the respective s-t-path and thus in general different for different leaves.

Example: assignment For PIAP-W+, branching can be based on vertices of the problem graph, similarly to PISPP-W+. For every vertex, exactly one arc incident has to be part of an assignment. The number of children of a branch-and-bound tree node is the degree of a vertex minus the number of already covered neighbours. The obtained branch-and-bound tree is in general not binary and has a maximal depth of half the number of vertices in the graph. However, it can be very wide if the graph is dense. Thus, a binary branching strategy on whether a single arc is used can also be advantageous.

Example: min cut Recall that in PIMCP-W+, there are two types of lower-level variables corresponding to the vertices and the arcs of the problem's graph. As a partial cut is given by two sets $J, \overline{J}$, it is straightforward to branch on the vertices. A vertex is assigned to either the side of the source or the sink. Variables corresponding to arcs can be fixed as soon as both end vertices are assigned. The resulting branch-and-bound tree is binary and has (without further considerations) a maximum depth equal to the number of vertices that are not assigned initially.

Alternatively, one could branch on whether an arc is forward-crossing the cut. However, the number of arcs is larger and fewer partially fixed configurations could be extended to a feasible solution.

4.1 Minimal lower-level completion

For obtaining a first incumbent as well as during the branch-and-bound search, we use a minimal lower-level complete solution that extends the partial required solution. We call y a *minimal lower-level completion* of $R \subset E$ if

$$y \in \arg\min_{y} \left\{ d^\top y : y \in Y, y_e = 1 \; \forall e \in R \right\} .$$

If the lower-level problem is given as a linear program, a minimal lower-level completion can be computed efficiently by adding the upper-level constraints on lower-level variables to the lower-level problem, reducing the set of feasible lower-level solutions:

$$\begin{array}{lrcll} \min\limits_{y} & d^\top y & & & \\ \text{s.t.} & y_e & = & 1 & \quad (\ e \in R\ ) \\ & Ay & = & b & \\ & y & \geq & 0. & \end{array}$$

In case there is no such y, the problem instance is infeasible. A minimal lower-level solution satisfying the upper-level constraints can also be evaluated by combinatorial methods based on the lower-level problem.

Example: shortest path For a shortest path in a directed acyclic graph, several subpaths are ordered by the topological order. A minimal completion is obtained by adding shortest paths (with respect to d without modifications) in the missing parts. There is at most one more missing part than the number of required arcs in the partial solution. This is the case if no required arc is incident to another required arc, the source or the sink.

Example: assignment In case the lower-level problem is an assignment problem, remove the already covered vertices from the graph. Solving the assignment problem in the remaining graph provides a minimal complete lower-level solution that extends the required partial solution.

Example: min cut If the underlying problem is a min cut problem, the required auxiliary graph is obtained by contracting the given subsets of vertices that are on either side. Afterwards, one evaluates a min cut in the remaining graph with respect to capacities d.

4.2 Primal bounding: complete ICP for minimal lower-level completion

Solving the complete ICP for a minimal lower-level completion turns out to yield a good initial upper-level solution for pruning at an early stage. A general LP-based method as well as combinatorial methods for the three example problems for evaluating a lower-level solution y^* are described in the previous Section 4.1. Remember that ICPs can be solved in polynomial time if the lower-level problem can be solved in polynomial time [2]. One can also use the described heuristic at any other branch-and-bound node than the root in order to obtain a primal bound.

4.3 Dual bounding

We propose two main strategies for obtaining dual bounds for a partial lower-level solution based on the two single-level reformulations via the KKT conditions respectively strong duality. The bounding strategy based on strong duality, described in Subsection 4.3.1, turns out to be good at the beginning, close to the root of the branch-and-bound tree. It is not necessarily tight for a complete lower-level solution. In contrast, the other bounding strategy is based on the KKT conditions and becomes tighter when the lower-level partial solution is close to a complete solution, see Subsection 4.3.1. For tree nodes where the required set is a subset of the minimal completion used for the initial solution, we will only try to bound with the first method described in the following.

4.3.1 Require lower-level objective to be at least value ℓ

Let ℓ be a valid lower bound on the *lower-level* objective value of any bilevel-feasible solution. We obtain a relaxation when requiring the lower-level objective value to be at least ℓ instead of fixing the required elements. In the following, we show that the relaxation can be evaluated as a single-level LP. Afterwards, we explain how the bound provided is globally valid and can be re-used for pruning throughout the branch-and-bound search.

If weights can only be increased, an ℓ as required can be obtained as the optimal value $\ell(R^{(i)})$ of a minimal lower-level completion of $R^{(i)}$. Hence, the optimal objective value $r(\ell)$ of the following bilevel problem yields a lower bound on the upper-level objective:

$$\begin{aligned} r(\ell) = \min_{w} \quad & \sum_{e \in E} w_e \\ \text{s.t.} \quad & 0 \leq w \\ & \ell \leq \min \left\{ (d+w)^\top y \mid Ay = b, y \geq 0 \right\}. \end{aligned} \tag{Req-ℓ-Relax}$$

Next, we show that this problem of the upper level requiring the lower-level objective to be at least ℓ can be solved as a single-level LP based on strong duality.

Theorem 4.1. *The lower bound $r(\ell)$ on the upper-level objective can be obtained by solving a single-level LP.*

Proof. Only the lower-level problem's objective in Program (Req-ℓ-Relax) is used. As the lower level is an LP, by strong duality, the dual problem has the same optimal objective value. Thus, we can replace the lower level by its dual $\max_\lambda \left\{\lambda^\top b \mid \lambda^\top A \leq d + w\right\}$. Observe that the dual objective is linear and only depends on the lower-level dual variables λ. The upper-level variables w are included in the constraints. The bilinear primal lower-level objective results in constraints that are linear in both the upper-level variables w and the lower-level dual variables λ.

Instead of explicitly solving for a maximal lower-level solution, we can combine the selection of the dual variables λ with the upper level choices. In total, we have

$$\begin{aligned} r(\ell) &= \min_{w} \left\{ \sum_e w_e \mid 0 \leq w,\ \ell \leq \min_y \left\{ (d+w)^\top y \mid Ay = b,\ y \geq 0 \right\} \right\} \\ &= \min_{w} \left\{ \sum_e w_e \mid 0 \leq w,\ \ell \leq \max_\lambda \left\{ \lambda^\top b \mid \lambda^\top A \leq d + w \right\} \right\} \\ &= \min_{w,\lambda} \left\{ \sum_e w_e \mid 0 \leq w,\ \ell \leq \lambda^\top b,\ \lambda^\top A \leq d + w \right\}. \end{aligned}$$

□

This bound can be reused for further partial solutions as it only depends on the value ℓ, and not the node ℓ was obtained from. First, by storing the tuple $(\ell, r(\ell))$ one can save the evaluation of the LP if the same ℓ is encountered for another node. Furthermore, the minimal required additional weight $r(\ell)$ is a non-decreasing function in ℓ. If for some node i with $\ell^{(i)}$ the value $r(\ell^{(i)})$ was already at least as large as the incumbent's objective value c such that i can be pruned, we can directly prune all other nodes i' with partial solutions with $\ell^{(i')} \geq \ell^{(i)}$ without evaluating the LP. Similarly, if a value $\ell^{(i)}$ did not suffice for pruning, the same will hold for all $\ell \leq \ell^{(i)}$, see Figure 7. Thus, we can save time and directly continue without evaluating the exact value of $r(\ell)$. For the latter strategy, we only need to save the smallest respectively largest value of ℓ such that $r(\ell)$ is at least as large respectively smaller than the current incumbent's value c, see Algorithm

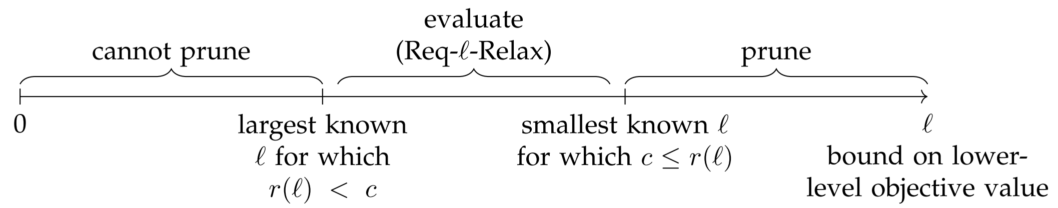
cannot prune
evaluate (Req-ℓ-Relax)
prune
0
largest known ℓ for which $r(\ell) < c$
smallest known ℓ for which $c \leq r(\ell)$
ℓ
bound on lower-level objective value

Figure 7: Strategy on whether to solve (Req-ℓ-Relax) based on newly encountered $\ell^{(i)}$. The function r for the minimal amount of required additional weight for inducing a lower-level solution of length $\geq \ell$ is nondecreasing. Breakpoints may shift left once a better incumbent is found. Furthermore, if pruning by $\ell^{(i)}$ fails, one may attempt to prune using an only locally valid ℓ.

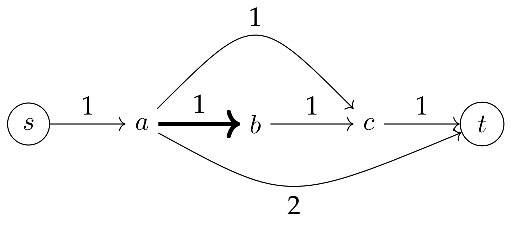

Figure 8: Require shortest path length bound not tight: For $R = \{ab\}$ the minimal completion has length $\ell = 4$. Requiring the shortest s-t-path to have length at least 4 has an optimal solution of only adding $w(sa) = 1$ which is less than the optimal objective value of $2 = 1 + 1 = w^*(ac) + w^*(at)$.

1. The use of this strategy can be increased by a node selection rule based on the minimal required objective value, see Section 4.5. If a small ℓ is known early, many branch-and-bound tree nodes can be pruned by only a value lookup.

This bound is not tight. For an example with the shortest path problem, see Figure 8.

Algorithm 1 Check-bound-by-min-completion-length(ℓ)

```
Require: Global variables $\ell_{branch}, \ell_{bound}$        ▷ Initialization $\ell_{branch} \leftarrow 0,\ \ell_{bound} \leftarrow \infty$,
                                                                  reused for subsequent calls
  if $\ell_{branch} < \ell < \ell_{bound}$ then                 ▷ Update $\ell_{bound}$ or $\ell_{branch}$
      if $c \leq r(\ell)$ then                                  ▷ (Req-$\ell$-Relax)
          $\ell_{bound} \leftarrow \ell$
      else
          $\ell_{branch} \leftarrow \ell$
  return $(\ell \geq \ell_{bound})$                             ▷ True or False
```

A tighter though not reusable version $r(\ell, R^{(i)})$ can be obtained by using in addition to the required minimal length ℓ the set of required elements $R^{(i)}$ for the current branch-and-bound node i. First, weights of elements in $R^{(i)}$ will not be increased in an optimal upper-level solution. Thus, the weight modifications for elements in $R^{(i)}$ can be fixed to 0. Furthermore, the corresponding dual constraints for these elements have to be fulfilled with equality due to complementarity. By adding these constraints to the LP reformulation for (Req-ℓ-Relax), we obtain the following stronger bound:

$$\begin{aligned} r(\ell, R^{(i)}) = \min_{w,\lambda} \quad & \sum_e w_e & & & \text{(Req-}(\ell, R^{(i)})\text{-Relax)} \\ \text{s.t.} \quad & \lambda^\top b & \geq & \ell & \\ & \lambda^\top A - w & \leq & d & \\ & \lambda^\top A_e - w_e & = & d_e & (\; e \in R^{(i)} \;) \\ & w_e & = & 0 & (\; e \in R^{(i)} \;) \\ & w & \geq & 0 & \end{aligned}$$

These two additional sets of constraints may lead to a tighter bound. Thus, forbidding a weight increase for required elements in (Req-ℓ-Relax) and requesting the corresponding dual constraints to be fulfilled with equality tightens the bound – at the cost of re-usability.

Example: shortest path The necessary minimal total weight modification for a shortest s-t-path having a length of at least ℓ is solvable as a (complete) inverse shortest path problem in an auxiliary graph. Add to a graph an additional s-t-path via an auxiliary node with length set to ℓ.

Example: min cut Similarly to requiring a shortest path to have a minimum length, it is possible to require a min cut to have at least some specified capacity by solving a complete inverse min cut problem in a modified graph. To this end, add a single arc from the original sink to a new sink with capacity ℓ. This new arc in the modified graph is the only forward-crossing arc of the cut with all original vertices on one side and the new sink on the other side. The PIMCP for this cut is equivalent to our problem of requiring a min cut in the original graph to have at least a capacity ℓ.

Example: assignment In contrast to the previous two examples, it is unclear how requiring a minimal perfect matching to be at least ℓ can be formulated as an inverse assignment problem.

4.3.2 LP-relaxation of single-level reformulation via KKT conditions

The LP-relaxation of the single-level reformulation based on the KKT conditions provides a lower bound on the required additional weights. The single-level reformulation via the KKT conditions has a nonlinear constraint for every lower-level variable, see (KKT-1level). The dual constraints are linear. The nonlinear constraints on complementary slackness simplify to linear constraints if the corresponding lower-level variables are fixed on the upper level. We drop all bilinear constraints of (KKT-1level) that correspond to unfixed lower-level variables. Then, we obtain the following linear relaxation that provides a lower bound for the overall solution

$$\begin{aligned} c_{relax}(R) = \min_{w,y,\lambda} \quad & \sum_{e \in E} |w_e| & & & \text{(LP-relax-KKT)} \\ \text{s.t.} \quad & \lambda^\top A & \leq & d + w & \\ & Ay & = & b & \\ & y_e & = & 1 & (\; e \in R \;) \\ & 0 & = & \lambda^\top A_e - d_e - w_e & (\; e \in R \;) \\ & l^w & \leq & w \quad \leq u^w. & \end{aligned}$$

If all lower-level variables are fixed (i.e. at the leaves of the branch-and-bound tree), this is a complete ICP. Hence, we will denote the solution value of (LP-relax-KKT) in case of a complete solution J also by $c_{\mathit{inv}}(J)$. Otherwise, it is a subproblem that can be interpreted as one or multiple complete ICPs restricted to subinstances. The optimality of these subproblems is necessary such that we obtain a lower bound. It is not sufficient in general, so the bound is not tight.

Example: shortest path The LP-relaxation of the KKT-reformulation is the inverse multi-path problem for inclusion-wise maximal required subpaths. Before explaining why this is the case, we describe the inverse multi-path problem.

The inverse multi-path problem is to find minimal weight modifications such that several given paths are shortest paths between the respective terminals. While sharing the weight modifications, the shortest path problems for paths $M_i = \{s^i, \dots, t^i\}, i \in [m]$ on the lower level are independent of each other. Considering the single-level reformulation via the KKT conditions, there are separate lower-level dual variables for every such path. The difference between any two dual variables is the minimal distance between the corresponding vertices. Along an arc this is at most the arc's length including a possibly non-zero modification. For the arcs chosen to be part of the respective path this has to be tight.

$$
\begin{array}{lrcll}
\min & \sum_{e \in E} w_e & & & \\
\text{s.t.} & d_{uv} + w_{uv} & \geq & x^i_v - x^i_u & (\ \forall (u,v) \in E, i \in [m]\ ) \\
 & d_{uv} + w_{uv} & = & x^i_v - x^i_u & (\ \forall (u,v) \in M_i, i \in [m]\ ) \\
 & x^i & \geq & 0 & (\ i \in [m]\ )
\end{array}
$$

Except for fixed offsets between the dual variables corresponding to different required paths, the minimal distances are the same as weight modifications are shared. Thus, the dual variables of several required paths can be combined to one set of distances. Combining, the obtained formulation is exactly the LP-relaxation of the KKT-reformulation of the PISPP.

Example: assignment In the single-level reformulation for the problem with an assignment problem on the lower level, there is a nonlinear constraint for every edge. For a chosen edge, this implies that the corresponding dual constraint has to be fulfilled while it simplifies to a true statement otherwise. Thus, the relaxation for the assignment problem corresponds to the inverse assignment problem on the subgraph induced by the vertices incident to edges in R.

Example: min cut In the LP-relaxation of the KKT-based reformulation for the min cut problem, a flow is evaluated such that the required forward-crossing arcs are saturated. The dual problem of the min cut problem is the max flow problem. The complementarity constraints fix that the forward-crossing arcs of the cut are saturated. If only weight increases are allowed this means that only capacities of arcs that do not cross the cut are increased. Increased capacities of other arcs can be necessary such that flow can reach these arcs from the source, respectively leave it towards the sink. On paths from the source to the sink without a fixed arc to be crossing the cut, there can be any feasible flow not necessarily saturating any arc. Already fixed nodes without a neighbour that is fixed to be on the other side of the cut have no effect on this lower bound.

4.4 Pruning by infeasibility: subtrees for non-extendable partial solutions

When detecting partial solutions $R^{(i)}$ that cannot be extended to feasible lower-level solutions irrespective of weight modifications, the corresponding subtree can be pruned. In general, one can try to evaluate a

Algorithm 2 Branch-and-bound for partial inverse combinatorial problems

```
$H \leftarrow \{R\}\,,\ \ell_{branch} \leftarrow 0,\ \ell_{bound} \leftarrow \infty$        ▷ Global variables
$c \leftarrow c_{inv}(\arg\min \{d^\top y : y \in Y,\ y_e = 1\ \forall e \in R\})$        ▷ Inverse problem for minimal completion
while $H \neq \emptyset$ do
    choose $R^{(i)} \in H$, remove $R^{(i)}$ from $H$
    $\ell \leftarrow \min \{d^\top y : y \in Y,\ y_e = 1\ \forall e \in R^{(i)}\}$        ▷ Minimal completion length
    if Check-bound-by-min-completion-length($\ell$) then        ▷ See Algorithm 1
        continue
    else if $R^{(i)}$ is complete lower-level solution then
        if $c_{inv}(R^{(i)}) < c$ then        ▷ Inverse Problem
            $c \leftarrow c_{inf}(R^{(i)}),\ \ell_{branch} \leftarrow 0$
    else if $R^{(i)}$ is subsolution of initial solution (min-completion) then
        branch on lower-level variables, add new children to $H$
    else if $c > r(\ell, R^{(i)})$ and $c_{relax}(R^{(i)}) < c$ then        ▷ (Req-($\ell, R^{(i)}$)-Relax), (LP-relax-KKT)
        branch on lower-level variables, add new children to $H$
return $c$
```

minimal completion according to Subsection 4.1 in every step. Depending on the lower-level problem, combinatorial properties can be exploited. Furthermore, based on upper-level constraints on weight modifications, further (partial) lower-level solutions can be infeasible. Problem-specific methods are mentioned as part of Section 4.7 on propagation.

4.5 Node selection: minimal completion value

The node selection is based on the minimal completion value of the lower level. Maximizing the number of trees that can be bounded by a simple dictionary lookup minimizes the number of LPs that need to be solved. If the minimal lower-level value ℓ of the considered tree node is at least the current bound ℓ_b, no further evaluation is required, see Section 4.3.1. Thus, the goal is to have ℓ_b as small as possible at an early stage. This is best achieved by choosing a tree node with best minimal completion value ℓ on the lower level.

4.6 Branching

We aim to identify problem-specific subsets of variables such that exactly one is set to one or only branch on a subset of the lower-level variables. In case of the shortest path or the assignment problem, a MIP solver should find these sets as well, while it is not that clear for the min cut problem. This reduces the size of the branch-and-bound tree in comparison to simply branching on every lower-level variable independently. Furthermore, we try to avoid the creation of branches that do not contain any feasible lower-level solutions.

Example: shortest path A simple path contains exactly one element from every inclusion-wise minimal cut between the path's source and sink. In particular, arcs that are in an inclusion-wise minimal cut together with some required arc of the current partial solution cannot be part of a feasible solution. Furthermore, if an arc is not part of any simple path between required parts of the partial solution, it cannot be part of a feasible solution. An easily identifiable set of arcs to branch on is the set of all out-/ingoing arcs of either the head or tail of a so far required subpath, the problem's source or sink. The respective out-/indegree of the respective vertex provides an upper bound for the number of children. By directly checking for the existence of a path to the next required part, the number of children can turn out to be smaller. The vertex to branch at can be chosen for example such that it maximizes or minimizes the number of children, prolongs a shortest or longest (w.r.t. to the original weight or the number of arcs) subpath, a subpath incident to the problem's source or sink or where a missing part is closed as soon as possible for at least one child. The choice of some of these rules can coincide for some instances.

Example: assignment For PIAP, we can branch at a vertex that is not covered yet. Every vertex is covered by exactly one edge in an assignment. Thus, taking the edges that are incident to some not-yet-covered vertex provides a set of variables from which always exactly one has to be used. Different rules to choose a free vertex to branch at are possible. Choosing a vertex of maximal degree with respect to the remaining free arcs results in many children. In contrast, only few children are obtained when taking a vertex of minimal degree with respect to free arcs. Other possible choices are with respect to the sum over weights of free incident arcs that is maximized or minimized.

Example: min cut Lower-level variables of the LP for the min cut problem describe the partition of vertices and indicate for arcs whether they are forward-crossing the cut. It suffices to only branch on variables that fix vertices to either side of the cut. Whether arcs forward-cross the cut is then given and should be explicitly fixed.

Any partition of the vertex set is a feasible cut. However, there might not be an upper-level solution when only increasing weights is allowed. As the capacity of a cut only depends on arcs forward-crossing the cut, bounding methods only consider these arcs. Thus, it is reasonable to branch such that more arcs have both end vertices assigned after every step. This is the case when branching is done at a vertex that is both a successor of a vertex in S and a predecessor of a vertex in $\overline{S}$. If no such vertex exists, at least in one of the two branches a new forward-crossing arc is fixed when we branch at a vertex that is either of the two.

4.7 Propagation

For a partial lower-level solution, all feasible completions might include certain further lower-level elements. Using these implications explicitly as soon as they are detected, we can prevent to solve subproblems at branch-and-bound tree nodes with a single child and reduce the height of the branch-and-bound tree. The following propagation methods are based on the specific lower-level problems.

Example: shortest path In PISPP, there is an upper-level solution for every simple path. Thus, we only need to consider whether a partial solution can be extended to a simple path. E.g., arcs that would create a cycle can be excluded directly. If there is only one possibility to continue a path because of a single out-/ingoing arc at a non-terminal node that is at the end/start of a partial path, the respective arc has to be included.

Example: assignment Similarly as for paths, there is an upper-level solution for every assignment. However, not every partial assignment can be extended to an assignment. If the induced subgraph of the not-yet-covered vertices has a connected component with an odd number of vertices, the instance is infeasible. In particular, this covers the case of an isolated vertex that cannot be completed to an assignment. This is not a sufficient condition. For example, there is no assignment in the complete bipartite graph $K_{2,4}$ though it has an even number of vertices. If there is a vertex with degree one in the induced subgraph of the not-yet-covered vertices, the respective incident arc has to be included in the partial solution.

Example: min cut In contrast to shortest path and assignment, every partial cut can be extended to a cut by any allocation of the remaining vertices to either side. However, for a min cut problem on the lower level with only weight increases, an upper-level solution might not exist. For every vertex $v \in S$ that is incident to a cut-crossing arc, v must be reachable from the source s within the set S. If there is a vertex $v \in S$ that is currently not reachable from s within S and there is only one incoming arc to v from a not assigned vertex $u \in V \setminus (S \cup \bar{S})$, then u needs to be part of S as well. For $\bar{S}$ an analogue argument with vertices from which the sink t is not reachable holds as well.

During the described branching, we only fix the variables corresponding to vertices. As soon as both end vertices of an arc have been assigned, it is fixed whether the arc is forward-crossing and thus the corresponding variable can be explicitly fixed.

5 Computational Results

5.1 Implementation, software and computing infrastructure

For our implementation we use Python (version 3.8.10) and Gurobi (version 10.0.2) as (I)LP-solver via the Python library gurobipy. Handling of graphs and some combinatorial subproblems are based on Networkx 3.1. Computations are run on an AMD EPYC 7742 64-Core Processor. For all computations a single core is allocated and the number of threads used by Gurobi is limited to one. This is done for reasons of fair comparison, since our prototype implementation is not parallelized,

Branch and bound We implement a basic branch-and-bound scheme in Python, solving the LPs with Gurobi via gurobipy. Minimal completions for the shortest path problem are evaluated with the bidirectional Dijkstra algorithm implemented in networkx. For underlying assignment or min cut problems, the according LP for minimal completion is solved with Gurobi, as well as all LPs for the dual bounds.

Single-level reformulation with indicator constraints The single-level reformulations are implemented with gurobipy using built-in indicator constraints. Preliminary experiments have shown that the reformulation based on strong duality can be solved significantly faster than the one based on complementarity constraints. Thus, we only consider the version via duality in the following.

Decomposition The decomposition approach is implemented in Python using gurobipy and Gurobi as solver for the master problem. For the assignment and the min cut problem, we use an implementation with Gurobi via gurobipy for solving the subproblems, while we use implementations of Dijkstra's algorithm provided in networkx for the shortest path problem. However, in comparison to the other two solution approaches, the decomposition approach performs poorly in preliminary tests if only a single lower-level solution is found in every iteration. For the shortest path problem, adding several feasible lower-level solutions at once determined by using Dijkstra's algorithm showed a significant advantage over adding only a single one a time. Thus, we only include a comparison to this superior approach for computations in case of the shortest path problem.

5.2 Instances

The used test instances are all randomly generated such that the required properties are fulfilled. In the following, we describe the used methods. The numbers of vertices respectively edges are input parameters to the generation of all instances. Weights are always chosen uniformly at random from the set $\{1, 2, ..., 1000\}$. As it turns out, performance of the three methods depends to a large degree on the density of the graph for partial inverse shortest path, assignment and min cut problems, so we discuss sets of test instances with varying density. Instances are available on request.

Shortest path For testing the different methods for PISPP, we randomly generate directed acyclic graphs. Given a number of vertices, we start with a path of according length and add shortcuts until the required number of edges is reached.

We use the following procedure with different random seeds: Independently choose two random vertices. If they are the same vertex or in case there is already an edge between these two vertices, repeat picking two vertices at random. Otherwise, add an arc according to the topological order defined by the path. Source and sink for the problem are the terminal vertices of the path. A randomly chosen arc from the graph is fixed as the unique required arc in R for the PISPP instance.

By construction, the topological ordering is unique. Furthermore, vertices close to the source have large out-degree and vertices close to the sink have large in-degree. The expected total degree is the same for all vertices.

Assignment We generate random bipartite graphs for computations for PIAP. For a given even number of vertices, fix a partition. Choose one vertex from each of the two sets uniformly at random and add an edge if it is not already present until the required number of edges is reached.

Min cut From a set of the corresponding number of vertices, two vertices are fixed as source and sink, respectively. For an arc we randomly choose a tail from all vertices except the sink and a head from all vertices except the source. As we do not want loops, we skip in case the same (non-terminal) vertex has been chosen. The same applies if there is already the according arc. Otherwise, the arc is added to a graph with a randomly chosen weight.

The constructed graph may consist of more than one connected component. In case the terminal vertices are in different connected components, we directly detect infeasibility of the instance and do not consider it further. Connected components without the terminal vertices are removed.

A random vertex is chosen to be fixed on either side of the cut to complete an instance for PIMCP.

5.3 Computations

For all three problems, we use a time limit of one hour for every instance and method. For instances that are not solved within the time limit, we use the time limit as running time for computing figures like mean or standard deviation.

5.3.1 Partial inverse shortest path

We consider instances with 1000 nodes, see Figure 9 respectively Table 2. In dense graphs, it turns out that the initial feasible solution found by our proposed heuristic via a minimal completion is often optimal. Furthermore, the method based on the minimal required length of a shortest path described in Section 4.3.1 provides in most cases directly the required bound to show optimality. In contrast, in sparse graphs, both the decomposition approach and Gurobi dealing with the single-level reformulation are superior. In case of the decomposition approach, each time the subproblem is solved, several optimal lower-level optimal solutions are evaluated. For graphs with an intermediate density, the running time for all three methods is rather large. In further experiments with larger instances, only sparse graphs could be solved by Gurobi and dense graphs could be solved with our branch-and-bound method.

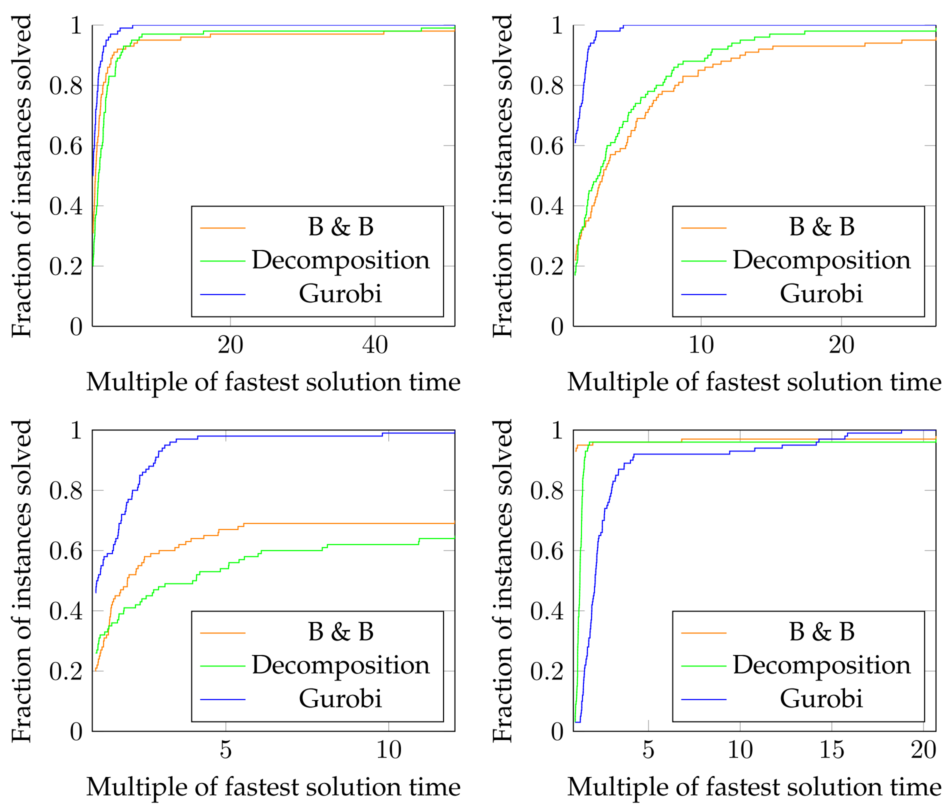

Figure 9: Performance for PISPP on instances with one required arc and graphs with 1000 vertices and 100 instances each with 50 000 arcs (upper left), 100 000 arcs, 200 000 arcs and 490 000 arcs (lower right). Time limit 1 hour per instance.

Which branching rule performs best mainly dependents on the instance type, i.e. the graph structure. For the described instances, the branching rules of prolonging the required path on the end where more arcs are incident worked best. As a result, a maximal number of children is created.

5.3.2 Partial inverse assignment

For PIAP, the success of our branch-and-bound method heavily depends on whether the bounds obtained at the root node already agree, and the result is quite lopsided either way: If the heuristic solution is optimal and the dual bound based on the minimal required size allows showing this, branch-and-bound (or rather: bound) is at least twice respectively three times as fast as Gurobi, see Figure 10. Otherwise, our own implemented branch-and-bound is not able to solve the problem to optimality within the time limit of one hour. However, some of these instances can be solved by Gurobi within the time limit. The maximal time needed by our method if it solves the instance is for all classes of graphs smaller than the according minimal time needed by Gurobi, see Table 3 for statistics on the running times of solved instances.

Denser graphs seem to be more difficult for PIAP. The number of solved instances decreases for both methods if instances have increasing density.

Number of edges	Method	Solved	Mean [s]	Std [s]	Min [s]	25% [s]	50% [s]	75% [s]	Max of solved [s]
50 000	B & B	98	181.1	578.8	8.2	20.8	35.9	78.9	2182.9
	Gurobi	100	45.3	45.8	14.1	20.8	26.3	49.6	301.4
	Decomposition	99	153.9	451.1	8.1	24.3	49.2	81.8	2458.1
100 000	B & B	95	583.3	831.7	17.6	68.2	287.9	671.3	2161.1
	Gurobi	100	108.4	89.3	30.6	63.1	80.2	110.4	635.1
	Decomposition	98	439.0	649.5	23.7	67.2	233.5	484.3	2983.6
200 000	B & B	69	1458.6	1572.3	26.1	42.7	499.5	3600.0	3264.1
	Gurobi	99	603.6	727.2	27.6	111.5	322.6	866.0	2953.4
	Decomposition	64	1544.9	1616.0	15.0	230.7	475.4	3600.0	2965.2
490 000	B & B	97	211.3	662.7	68.5	73.1	75.7	79.8	2913.6
	Gurobi	100	289.3	381.4	67.8	131.2	165.8	235.2	2729.6
	Decomposition	96	237.0	690.1	40.3	84.3	96.7	106.7	145.1

Table 2: Statistics on running times (in seconds) for PISPP as in Figure 9

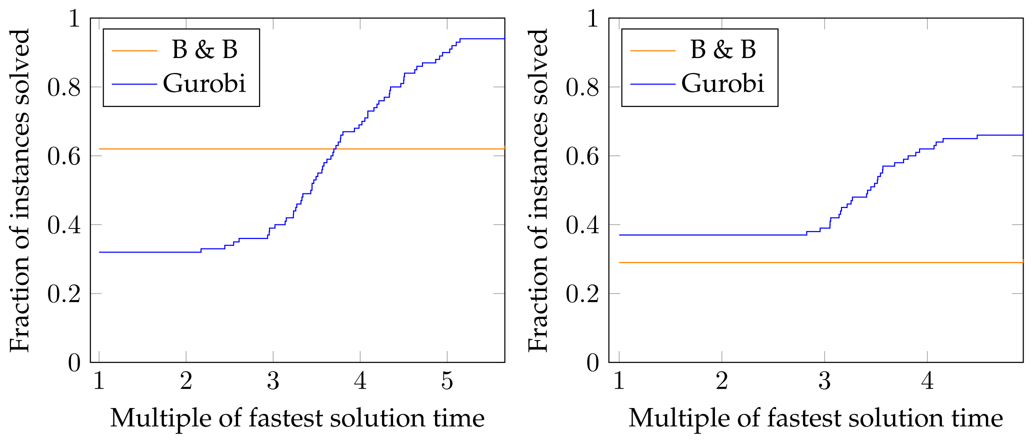

Figure 10: Performance for PIAP on instances with 3000 vertices, (all together), 50 000 (left) respectively 100 000 edges, 1h time

Number of edges	Method	Solved	Mean [s]	Std [s]	Min [s]	25% [s]	50% [s]	75% [s]	Max of solved [s]
50 000	B & B	62	1382.0	1745.2	13.7	14.1	32.8	3600.0	45.4
	Gurobi	94	368.9	889.0	48.4	66.1	99.9	145.1	3271.2
100 000	B & B	29	2574.1	1613.4	53.1	67.3	3600.0	3600.0	75.8
	Gurobi	66	1542.8	1555.6	174.6	242.1	484.6	3600.0	3248.3

Table 3: Statistics on running times (in seconds) for PIAP as in Figure 10

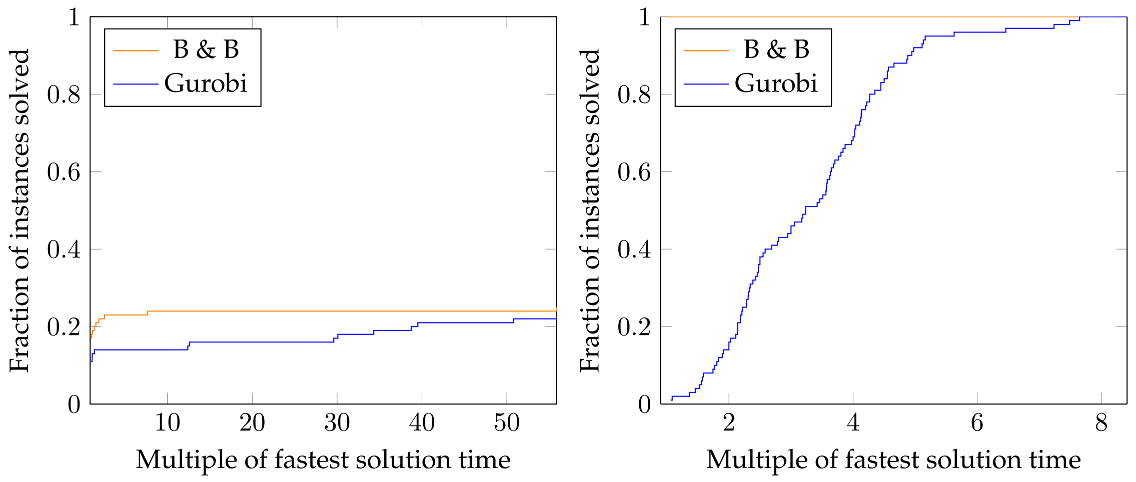

Figure 11: PIMCP - sparse vs dense graph: 100 instances, each with 150 vertices, 10 000 arcs (left) respectively 220 000 arcs, time limit of 1 hour (though 1 min suffices in all cases of 2.2e5 arcs, see Table 4)

Number of arcs	Method	Solved	Mean [s]	Std [s]	Min [s]	25% [s]	50% [s]	75% [s]	Max of solved [s]
10 000	B & B	24	2801.7	1456.4	1.0	3600.0	3600.0	3600.0	2769.8
	Gurobi	22	2896.4	1371.0	1.0	3600.0	3600.0	3600.0	3468.6
220 000	B & B	100	2.9	0.3	2.3	2.6	2.8	3.0	3.8
	Gurobi	100	11.9	5.5	2.5	7.9	11.5	15.9	28.2

Table 4: Statistics on running times (in seconds) for PIMCP as in Figure 11

5.3.3 Partial inverse min cut

Similarly to partial inverse shortest path, our prototype branch-and-bound implementation outperforms Gurobi on dense graphs, see Figure 11. However, the amount of time needed by any method is quite small with less than one minute for dense graphs, see Table 4. In contrast, only few sparse graphs are solved by either method within a time limit of one hour per instance. For those instance solved faster by our own branch-and-bound method, the factor of time needed by Gurobi can be quite large in some cases.

6 Conclusion

This paper proposes one of the first algorithmic contributions to solving $\mathcal{NP}$-complete PICPs in which weights can only be increased. Our algorithm uses a branch-and-bound scheme with novel primal and dual bounding procedures that exploit the specific structure of these problems – in particular that problems resembling complete ICPs can be solved efficiently. For three example test cases using shortest path, assignment and minimum cut problems on the lower level, our prototype implementation is already competitive with a state-of-the-art solver applied to a single-level reformulation, even though it is implemented in python and doesn't enjoy access to all the features of a mature solver. The computational study suggests that the strengths of both methods are somewhat supplementary. In case of underlying shortest path or min cut problem instances, dense graphs favour our branch-and-bound implementation while Gurobi solving the single-level reformulation is superior on sparse graphs. For PIAP, our heuristic in combination with the dual bound performs much better for dense graphs. It is not unusual for our method to solve an instance right at the root node or shortly afterwards, resulting in a large speed-up. That means that a combination might yield a significantly more powerful method, e.g. by integrating our bounding subproblems into Gurobi (or any other suitable solver). Indeed, we observed that already the bounds we generate at the root node can be hard for Gurobi to reproduce for certain instances and might be very valuable.

In the paper we contributed two complexity results on versions of PICP being $\mathcal{NP}$-complete. For such problems one can, however, ask for milder conditions than completeness of the solution provided by R on when the problem becomes solvable in polynomial time. Such insights may be used to guide branching decisions. For example, is PISPP-W+ still $\mathcal{NP}$-complete if all of R is connected to the source or sink? If the answer was no, one consequence could be that branching should aim to achieve this quickly. Finally, one could try to extend our algorithmic ideas to PICPs with more difficult combinatorial problems on the lower level.

References

[1] Sara Ahmadian, Umang Bhaskar, Laura Sanità, and Chaitanya Swamy. Algorithms for inverse optimization problems. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 1:1–1:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[2] Ravindra K. Ahuja and James B. Orlin. Inverse optimization. *Operations Research*, 49(5):771–783, 2001.

[3] Ravindra K. Ahuja and James B. Orlin. Combinatorial algorithms for inverse network flow problems. *Networks*, 40(4):181–187, 2002.

[4] Merve Bodur, Timothy C. Y. Chan, and Ian Yihang Zhu. Inverse mixed integer optimization: Polyhedral insights and trust region methods. *INFORMS Journal on Computing*, 34(3):1471–1488, 2022.

[5] Aykut Bulut and Ted K. Ralphs. On the complexity of inverse mixed integer linear optimization. *SIAM Journal on Optimization*, 31(4):3014–3043, 2021.

[6] Mao-cheng Cai, C. W. Duin, Xiaoguang Yang, and Jianzhong Zhang. The partial inverse minimum spanning tree problem when weight increase is forbidden. *European Journal of Operational Research*, 188(2):348–353, 2008.

[7] Herminia I. Calvete and Carmen Galé. *Algorithms for Linear Bilevel Optimization*, pages 293–312. Springer International Publishing, Cham, 2020.

[8] Marc Demange and Jérôme Monnot. *An Introduction to Inverse Combinatorial Problems*, chapter 17, pages 547–586. John Wiley & Sons, Ltd, Hoboken/London, 2014.

[9] Scott DeNegre. *Interdiction and Discrete Bilevel Linear Programming*. PhD thesis, Lehigh University, 2011.

[10] Robert B. Dial. Network-optimized road pricing: Part I: A parable and a model. *Oper. Res.*, 47(1):54–64, 1999.

[11] Matteo Fischetti, Ivana Ljubic, Michele Monaci, and Markus Sinnl. A new general-purpose algorithm for mixed-integer bilevel linear programs. *Operations Research*, 65(6):1615–1637, 2017.

[12] Matteo Fischetti, Ivana Ljubic, Michele Monaci, and Markus Sinnl. Interdiction games and monotonicity, with application to knapsack problems. *INFORMS J. Comput.*, 31(2):390–410, 2019.

[13] Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.

[14] Fabio Furini, Ivana Ljubic, Sébastien Martin, and Pablo San Segundo. The maximum clique interdiction problem. *European Journal of Operational Research*, 277(1):112–127, 2019.

[15] Elisabeth Gassner. The partial inverse minimum cut problem with L_1-norm is strongly NP-hard. *RAIRO - Operations Research*, 44(3):241–249, 2010.

[16] Clemens Heuberger. Inverse combinatorial optimization: A survey on problems, methods, and results. *Journal of Combinatorial Optimization*, 8(3):329–361, 2004.

[17] Eitan Israeli and R. Kevin Wood. Shortest-path network interdiction. *Networks*, 40(2):97–111, 2002.

[18] Thomas Kleinert, Martine Labbé, Ivana Ljubic, and Martin Schmidt. A survey on mixed-integer programming techniques in bilevel optimization. *EURO Journal on Computational Optimization*, 9:100007, 2021.

[19] Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer, Berlin, 6 edition, 2018.

[20] Tsung-Chyan Lai and James Orlin. The complexity of preprocessing. Technical report, Sloan School of Mangement, 11 2003.

[21] Xianyue Li, Zhao Zhang, and Ding-Zhu Du. Partial inverse maximum spanning tree in which weight can only be decreased under $\mathrm{l_p}$-norm. *Journal of Global Optimization*, 70(3):677–685, 2018.

[22] Churlzu Lim and J. Cole Smith. Algorithms for discrete and continuous multicommodity flow network interdiction problems. *IIE Transactions*, 39(1):15–26, 2007.

[23] Ayalew Getachew Mersha and Stephan Dempe. Linear bilevel programming with upper level constraints depending on the lower level solution. *Applied Mathematics and Computation*, 180(1):247–254, 2006.

[24] T. J. Moser. Shortest path calculation of seismic rays. *Geophysics*, 56(1):59–67, 01 1991.

[25] J. Cole Smith and Yongjia Song. A survey of network interdiction models and algorithms. *European Journal of Operational Research*, 283(3):797–811, 2020.

[26] Lizhi Wang. Cutting plane algorithms for the inverse mixed integer linear programming problem. *Operations Research Letters*, 37(2):114–116, 2009.

[27] Ningji Wei and Jose L. Walteros. Integer programming methods for solving binary interdiction games. *European Journal of Operational Research*, 302(2):456–469, 2022.

[28] Xiaoguang Yang. Complexity of partial inverse assignment problem and partial inverse cut problem. *RAIRO - Operations Research*, 35(1):117–126, 2001.

[29] Xiaoguang Yang and Jianzhong Zhang. Partial inverse assignment problems under l_1 norm. *Operations Research Letters*, 35(1):23–28, 2007.

[30] Rico Zenklusen. Matching interdiction. *Discrete Applied Mathematics*, 158(15):1676–1690, 2010.

[31] Zhao Zhang, Shuangshuang Li, Hong-Jian Lai, and Ding-Zhu Du. Algorithms for the partial inverse matroid problem in which weights can only be increased. *Journal of Global Optimization*, 65(4):801–811, 2016.