Ley, Eva; Merkert, Maximilian

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

Journal of Global Optimization

Provided in Cooperation with:
Springer Nature

Suggested Citation: Ley, Eva; Merkert, Maximilian (2025) : Solution methods for partial inverse combinatorial optimization problems in which weights can only be increased, Journal of Global Optimization, ISSN 1573-2916, Springer US, New York, NY, Vol. 93, Iss. 1, pp. 263-298, https://doi.org/10.1007/s10898-025-01529-x

This Version is available at:
https://hdl.handle.net/10419/330785

Journal of Global Optimization (2025) 93:263–298
https://doi.org/10.1007/s10898-025-01529-x

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

Eva Ley[1] · **Maximilian Merkert**[1]

Received: 8 February 2024 / Accepted: 26 July 2025 / Published online: 27 August 2025

Abstract
Partial inverse combinatorial optimization problems are bilevel optimization problems in which the leader aims to incentivize the follower to include respectively not include given sets of elements in the solution of their combinatorial problem. If the sets of required and forbidden elements define a complete follower solution and the follower problem is solvable in polynomial time, then the inverse combinatorial problem is also solvable in polynomial time. In contrast, partial inverse problems can be NP-complete when the follower problem is solvable in polynomial time. This applies e.g. to the partial inverse min cut problem. 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, we show that the partial inverse shortest path problem on a directed acyclic graph is NP-complete. Moreover, the partial inverse assignment problem is NP-complete. Both results even hold if there is only one required arc or edge, respectively. 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, we use auxiliary problems that are basically complete inverse problems on similar instances. 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 · 90C27 · 90C26 · 90C60 · 91A65

✉ Maximilian Merkert
m.merkert@tu-braunschweig.de

Eva Ley
eva.ley@tu-braunschweig.de

[1] Institute for Mathematical Optimization, TU Braunschweig, Universitätsplatz 2, 38106 Braunschweig, Lower Saxony, Germany

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 there is a lower-level optimal solution extending a given partial lower-level solution. In a partial solution of a combinatorial problem some elements are required to be included while others are forbidden. It is, thus, a bilevel optimization problem with bound constraints on lower-level variables on the upper level. PICPs occur e.g. in economics when a planner wants rational agents to behave in a prescribed way, see e.g. [1]. A PICP can also model parameter estimation problems, in which one aims to infer parameter values that explain a partially observed action, see e.g. [2].

If the given lower-level solution is a complete solution, we can solve the resulting *complete* inverse combinatorial problem (ICP) in polynomial time if the lower-level problem itself is polynomial-time solvable [3]. However, as soon as the target solution is only partial, the *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. For example, the partial inverse min cut problem (PIMCP) is $\mathcal{NP}$-complete [4].

Throughout most of this paper, we assume that the lower-level problem admits an LP formulation and is therefore solvable in polynomial time. Under this assumption, once we know a lower-level feasible solution that extends the given partial solution and is optimal on the lower level in an optimal solution of the PICP, we can compute optimal weight modifications for the PICP efficiently by solving the corresponding complete ICP. We call a lower-level feasible solution that extends the given partial lower-level solution a *lower-level completion*. Hence, the main challenge in solving a PICP can be seen in determining which lower-level completion will be optimal for the follower in an optimal solution of the PICP.

In some cases, we can optimally solve a PICP in polynomial time by solving the corresponding complete ICP of a lower-level completion that is minimum-weight with respect to the initial weights. We call such a lower-level completion a *minimum-weight lower-level completion*, or just briefly *minimal completion*. Examples for PICPs that are solvable as complete ICPs for a minimal completion are the partial inverse shortest path problem (PISPP) on directed acyclic graphs (DAGs), see Theorem 2.11, and the partial inverse assignment problem (PIAP) [5], if arbitrary weight modifications are allowed.

However, if we impose further restrictions on the weight modifications, the minimal-completion strategy can fail. We denote a PICP with only weight increases by PICP-W+. For PISPP-W+ on a DAG, an optimal solution of the complete ICP for a minimal completion can be arbitrarily worse than an optimal solution of the PISPP-W+, see Figure 1. In fact, we show that both PISPP-W+ on DAGs and PIAP-W+ are $\mathcal{NP}$-complete, see Theorem 3.1 and Theorem 3.2, respectively.

Our main contribution is a novel branch-and-bound method for $\mathcal{NP}$-complete variants of PICP-W+ whose lower-level problem is solvable as an LP. Both for primal and dual bounding, we exploit that the corresponding *complete* ICPs are solvable in polynomial time. For primal bounds, we use the complete ICP for a minimal completion. Although this minimal-completion-based heuristic can be arbitrarily far from optimal in the worst case (as shown by the example from Figure 1), it yields very strong incumbents in practice. For dual bounds, we derive two relaxations that closely resemble *complete* ICPs. For PISPP-W+ on a DAG and PIMCP-W+, we can even model these constructions explicitly as corresponding ICPs.

In a computational study on three example problem classes, a prototype implementation of our branch-and-bound overall outperforms a state-of-the-art solver on a standard single-level

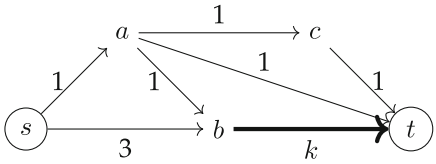

Fig. 1 Example for the partial inverse shortest path problem with only weight increases (PISPP-W+) with required elements $R = \{(b, t)\}$, no forbidden elements $F = \emptyset$ and parameter $k \geq 3$: The shortest s-t-path with respect to the initial weights is $\{(s, a), (a, t)\}$. There are two s-t-paths containing the required arc (b, t): the path $\{(s, b), (b, t)\}$ and the path $\{(s, a), (a, b), (b, t)\}$. The latter is (w.r.t. the initial weights) the shortest s-t-path containing the required arc (b, t). It is thus the *minimum lower-level completion* (see Definition 2.3). PISPP-W+ asks for minimal weight increases (measured in the ℓ_1-norm) such that there is a shortest s-t-path including the required arc (b, t). A minimal weight increase such that the minimum lower-level completion $\{(s, a), (a, b), (b, t)\}$ is a shortest s-t-path, is to add k to the weight of arc (a, t) and $k - 1$ to the weight of (a, c) or (c, t), i.e. a total modification of $2k - 1$. In contrast, we only need to add $k + 1$ to the weight of arc (s, a) for the path $\{(s, b), (b, t)\}$ to become a shortest s-t-path. The optimal solution value to this PISPP-W+ instance is thus $k + 1$. Note that it is not the minimal completion that became a shortest path in the optimal solution. Furthermore, by increasing the parameter k we see that the gap between the optimal solution value and a solution of the complete inverse shortest path problem for a minimal completion can be arbitrarily large

reformulation. As examples, we consider three classical problems: the shortest path problem on a directed acyclic graph (DAG), the assignment problem and the min cut problem. In addition, we close some gaps in the complexity landscape by proving $\mathcal{NP}$-completeness of PISPP-W+ on a DAG, PIAP-W+ and PIMCP-W+.

We work with the *optimistic assumption*, i.e. whenever the follower is indifferent among multiple optimal solutions, they choose the one most favourable to the leader. For optimistic PICPs, it suffices that at least one lower-level optimal solution meets the leader's requirements. In contrast, for pessimistic PICPs every lower-level optimal solution must satisfy the upper-level requirements. Algorithmically, however, solving the optimistic and pessimistic versions of a PICP is essentially the same, see Section 2.7.

In Section 2, we formally define PICPs, derive optimality conditions, and introduce the specific problem variants we study. Furthermore, we review the related literature, outline general solution approaches, and briefly discuss the pessimistic version of PICPs. Afterwards in Section 3, we prove $\mathcal{NP}$-completeness for PISPP-W+, PIAP-W+ and PIMCP-W+. In Section 4, we present our branch-and-bound method for PICP-W+, with an emphasis on primal and dual bounding procedures. We also briefly discuss other aspects. In Section 5, we report computational results for three example classes of PICP-W+. Section 6 concludes with a summary of our findings.

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:

$$
\begin{array}{lrcll}
\min\limits_{w,y} & \sum\limits_{e\in E} |w_e| & & & \\
\text{s.t.} & l^w & \leq & w \leq u^w & \\
& y_e & = & 1 & (e \in R) \\
& y_e & = & 0 & (e \in F) \\
& y & \in & \arg\min\limits_{y'\in Y} (d+w)^\top y', &
\end{array}
$$

with a combinatorial optimization problem on the lower level. The lower-level feasible set $Y \subseteq \{0,1\}^n$ is independent of the upper-level variables w that modify the lower-level objective function $(d+w)^\top y$. We refer to d as initial weights. By slight abuse of notation, we identify the lower-level variable y both with the characteristic vector and the set $\{e \in E \mid y_e = 1\}$.

The leader must ensure that at least one optimal solution of the follower includes all required elements from a *set of required elements* R while avoiding any elements in a *set of forbidden elements* F. If R and F determine a unique lower-level solution, the problem is in fact a *complete* ICP. Unless stated otherwise, we assume that there are no forbidden elements, i.e. $F = \emptyset$.

We measure the weight modifications by the ℓ_1-norm, i.e. minimize $\sum_e |w_e|$ on the upper level. For some applications, a weighted sum might be more appropriate. Our algorithmic results extend to the weighted case with only minor adjustments.

For the upper-level bounds $l^w \leq w \leq u^w$, we can assume $l^w \leq 0 \leq u^w$ (possibly $\mp\infty$) by shifting the initial weights d and adding a constant to the leader's objective. If $l^w = 0$, we can only increase weights and denote this by the suffix 'W+'. Similarly, if $u^w = 0$, we have a PICP in which weights can only be decreased, and denote this by the suffix 'W-'.

2.2 Minimal lower-level completion

A *lower-level completion* is a feasible lower-level solution that includes all required elements R and excludes all forbidden elements F. Among these completions, those that minimise the initial weight $d^\top y$ – so-called *minimal completions* – are especially useful in our branch-and-bound scheme. Moreover, the existence of a lower-level completion provides an immediate feasibility test for a PICP instance.

Observation 2.1 *If no lower-level completion exists, the PICP problem instance is infeasible.*

Note that the reverse implication is false for general weight modification bounds l^w, u^w. For example, if no weight modifications are allowed ($l^w = u^w = 0$), then $w = 0$ might be infeasible – even though a lower-level completion exists. However, the reverse direction of Observation 2.1 holds for arbitrary weight decreases or increases, with additional assumptions:

Theorem 2.2 *In the following cases, the existence of a lower-level completion is equivalent to feasibility:*

1. *Let $F = \emptyset$. Then, a PICP-W- instance is feasible if and only if it has a minimal lower-level completion.*
2. *Let $R = \emptyset$. Then, a PICP-W+ instance is feasible if and only if it has a minimal lower-level completion.*

3. *Let either $F = \emptyset$ or $R = \emptyset$. Then, a PICP instance is feasible if and only if it has a minimal lower-level completion.*
4. *Assume that there is a lower-level completion that is not a proper subset of any other lower-level feasible solution. Then, a PICP, PICP-W- or PICP-W+ instance is feasible if and only if it has a minimal lower-level completion.*

Proof By Observation 2.1, the existence of a lower-level completion is necessary. We describe weight modifications such that a fixed lower-level completion becomes an optimal lower-level solution:

1. Decreasing arbitrarily all weights of elements in R will eventually yield a bilevel-feasible solution.
2. Increasing arbitrarily all weights of elements in F will eventually yield a bilevel-feasible solution.
3. Depending on whether $F = \emptyset$ or $R = \emptyset$, we obtain a bilevel-feasible solution by applying the strategy from either the first or the second case above.
4. Decreasing arbitrarily all weights of elements in the lower-level completion (in case of PICP or PICP-W-) or increasing arbitrarily all weights of elements that are not in the lower-level completion (in case of PICP or PICP-W+) will eventually yield a bilevel-feasible solution. □

The condition in the last part of the previous theorem is necessary if there are both required and forbidden elements and weights can only be either increased or decreased. To see this, let y^* be a bilevel-feasible lower-level solution that strictly contains another lower-level feasible solution $\bar{y}$. If there is a required element in $y^* \setminus \bar{y}$ with positive weight in a PICP-W+ instance, this instance can be infeasible. Similarly, a lower-level feasible solution $\tilde{y}$ that is a superset of y^* and contains a forbidden element with negative weight, can lead to an infeasible PICP-W- instance.

We obtain a lower-level completion by solving the underlying combinatorial problem on a modified instance where elements are fixed according to R and F. Whenever this problem type is solvable in polynomial time, Theorem 2.2 implies that we can test feasibility of a PICP instance in polynomial time.

Definition 2.3 A *minimal (lower-level) completion* y of required elements R and forbidden elements F in a PICP is a lower-level solution y that includes the required elements R, does not contain any forbidden elements from F, and is optimal if there is no weight modification (i.e. $w = 0$), i.e. a solution to

$$\min_y \left\{ d^\top y : y \in Y, y_e = 1 \; \forall e \in R, y_e = 0 \; \forall e \in F \right\}.$$

The *minimal completion value* $\ell(R, F)$ is the optimal objective value of this optimization problem.

Note that by the term *minimal completion* we always refer to an optimal lower-level completion *with respect to the initial weights*.

Observation 2.4 *The minimal completion value is a lower bound on the lower-level objective value in any bilevel-feasible solution if weights cannot be decreased.*

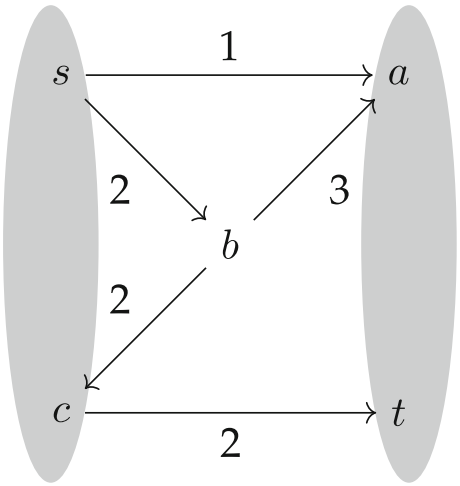

Fig. 2 Example for the partial inverse min cut problem (PIMCP) with $R = \{s, c\}, F = \{a, t\}$, i.e. we want to find minimal modifications (with respect to the ℓ_1-norm) of the capacities such that there is a min s-t-cut with vertex c on the same side as the source s and vertex a on the side of the sink t. A min s, t-cut w.r.t. the initial capacities is $(\{s, a, b, c\}, \{t\})$ with capacity 2. There are two possible lower-level completions: The cut $(\{s, c, b\}, \{a, t\})$ has an initial capacity of 6 and the cut $(\{s, c\}, \{a, t, b\})$ has an initial capacity of 5. Hence, the minimal completion value is $\ell(\{s, c\}, \{a, t, b\}) = 5$. For the former cut $(\{s, c, b\}, \{a, t\})$ to become a min s-t-cut, it is optimal to decrease the capacities of both (s, a) and (b, a) to 0 which is a modification by in total 4 units. The latter cut $(\{s, c\}, \{a, t, b\})$ becomes a min s, t-cut if the capacities of the three arcs (s, a), (s, b) and (c, t) are decreased to 0 which is a total modification by 5 units. Inequality 3 in Lemma 2.5 is strict in this example: $\sum_e |w_e^*| = 4 > 3 = 5 - 2 = \ell(R, F) - \min_y \left\{ d^\top y \mid y \in Y \right\}$. If we only allow weight increases, the instance becomes infeasible: In both possible lower-level completions, there is no a-t-path within the sink side. As long as all arcs have positive capacity, none of these lower-level completions can be optimal for the follower

2.3 Optimality conditions

We give some necessary optimality conditions for PICP that are a generalization of [5, Lemma 1].

Lemma 2.5 *Let (w^*, y^*) be an optimal solution of a PICP. 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. $\sum_e |w_e^*| \geq \ell(R, F) - \min_y \left\{ d^\top y \mid y \in Y \right\}$

Inequality 3 in Lemma 2.5 can be strict as the example for PIMCP in Figure 2 shows. However, inequality 3 is satisfied with equality if the lower-level problem is a linear program (LP) where arbitrary weight modifications are allowed for all variables, e.g. PIAP [5, Theorem 3].

2.4 Easy lower-level problems

We consider PICPs with lower-level problems that can be solved as a linear program (LP).

Assumption 2.6 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.

Based on this assumption, we have the following problem formulation, where we use equality constraints and non-negative variables on the lower level.

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

In this setting, we can find a minimal lower-level completion by solving a single-level LP (by fixing the y-variables for the required elements R and forbidden elements F). Moreover, when R and F specify a complete lower-level solution, the ICP is solvable as a single-level LP [3]. We obtain this single-level program by either using strong duality or complementarity/KKT conditions, see Section 2.6.

When formulating a combinatorial problem as an LP in standard form, auxiliary variables might be necessary that are not included in the LP's objective. In the corresponding PICP, no weight modifications for these additional variables are allowed. For example, when formulating the min cut problem as an LP, the variables corresponding to vertices do not appear in the objective.

Next, we describe PIAP, PISPP on a DAG and PIMCP as three classes of PICPs. We use these three example problem classes for complexity results in Section 3, problem-specific details for our branch-and-bound method in Section 4 and computational results in Section 5.

Example: assignment The assignment problem asks for 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 satisfies Assumption 2.6, see e.g. [6, Theorem 11.4]. Note that all variables y_e appear in the objective. The partial inverse assignment problem (PIAP) is to find minimal weight modifications such that there is a minimal assignment containing and not containing, respectively, accordingly specified edges. To get a minimal lower-level completion for PIAP, we first create an auxiliary graph by removing forbidden edges and nodes incident to required edges. Then, on this smaller graph, we find a minimal assignment with respect to the initial weights d. The union of such an assignment with the required edges is a minimal lower-level completion for PIAP.

Example: shortest path The shortest path problem asks for a minimum-cost path from a source s to a sink t in a directed graph. We formulate it as an LP fulfilling Assumption 2.6 by sending one unit of flow from the source to the sink while enforcing flow conservation at all non-terminal vertices. In the partial inverse shortest path problem (PISPP), the leader has to find minimal weight modifications such that there is a shortest s-t-path that contains all required arcs while avoiding forbidden arcs.

The flow-based LP formulation suggested above can be problematic if there are cycles in the graph. Required arcs may only be part of a solution within a 0-weight cycle in the

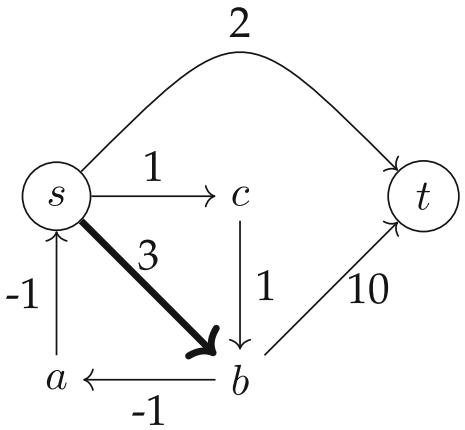

Fig. 3 Example for PISPP with $R = \{(s, b)\}$, $F = \emptyset$: The shortest s-t-path without weight modifications has length two. The minimal completion length is $\ell(R, F) = 13$. Reducing only the weight of the required arc (s, b) does not give a bilevel-feasible solution due to the negative weights on arcs (b, a), (a, s). The reduction of the weight of arc (s, b) would need to be at least $2 - 13 = -11$, resulting in a negative cycle through s, b, a. Only reducing the weight of arc (b, t) is not a solution either. An optimal solution is to reduce the weight of arc (s, b) by one and decrease the weight of arc (b, t) by 10 to 0

above LP formulation. This does not solve PISPP as a path is connected and visits every node at most once. Further issues due to negative cycles can occur, in particular if only weight decreases are allowed, see Figure 3. Hence, for simplicity we add the following assumption.

Assumption 2.7 For a shortest path problem on the lower level, we assume that the considered graph is acyclic.

For a minimal lower-level completion, we connect the source to the tail of the first required arc, each head of a required arc to the tail of the next required arc and the head of the last required arc to the sink by (w.r.t. d) shortest paths avoiding any forbidden arcs. In a directed acyclic graph (DAG), the order of the required arcs is given by a topological order. For obtaining a minimal lower-level completion, we need at most $|R| + 1$ shortest path computations.

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 2.8 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 has variables y_{uv} for all arcs and variables z_v for all vertices. It fulfills Assumption 2.6. 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} specifies whether the arc is forward-crossing the cut and hence needs to be counted.

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

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 and the sink, respectively. This is the definition used in [4] where this problem is shown to be $\mathcal{NP}$-hard. 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 of the z-variables are fixed on the upper level, they are not part of the lower-level objective. Instead, the lower-level objective only depends on the y-variables, i.e. the variables corresponding to arcs. Weight modifications for the vertex variables z have to be explicitly set to zero. Thus, we have $u^w \neq \infty$ for PIMCP-W+. Hence, the existence of a lower-level completion does not imply feasibility, see Theorem 2.2 and the example in Figure 2.

In a more general definition of PIMCP, one may also allow to fix whether certain arcs are forward-crossing the cut. We can require an arc to be forward-crossing in the above definition, by fixing its' end vertices accordingly. However, there are three different possibilities to fix the vertices of an arc that shall not be forward-crossing.

We obtain a minimal lower-level completion for PIMCP via a min cut (w.r.t. d) in an auxiliary graph where the fixed subsets of vertices are contracted. An example instance where solving an ICP for a minimal completion does not lead to an optimal solution of PIMCP is given in Figure 2.

2.5 Existing literature

Related problems

A PICP is a bilevel problem with *coupling constraints*. These are constraints on the upper level that explicitly depend on lower-level variables, see e.g. [7]. Alternative names for coupling constraints are *connecting constraints*, cf. [8], or *joint constraints*, cf. [9]. Coupling constraints pose several difficulties [7, 8]. 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 cannot handle coupling constraints, see [9]. The algorithm proposed by Fischetti et al. in [10] 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.

PICPs share key structural features with *interdiction* problems (or 'interdiction games'). In interdiction problems the leader prohibits or penalizes certain structures of the follower so as to worsen the follower's optimal value. There are two versions of interdiction problems. Either the leader has a budget for interdiction or a target for the lower-level objective. In 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. Both in the target version of interdiction problems and 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. In contrast to PICPs, in interdiction problems the leader variables are typically binary. Allowing continuous interdiction seems to be more challenging [11]. The leader's decision modifies the lower-level feasible set, or a large extra-cost is added to interdicted elements in the lower-level objective function [12]. In the latter case, the problem's structure is similar to the one of PICPs. Interdiction problems are a particularly well-studied subclass of bilevel problems and several specialized methods have been developed for them. They can be solved via decomposition approaches that are strengthened by cuts based on the lower-level problem, e.g. [13–15]. Problems that have been considered as lower-level problem for interdiction include both problems solvable in polynomial time like shortest path [12] and matching [16], as well as $\mathcal{NP}$-complete problems like knapsack [13] or max clique [14]. For further references also see [7, 15, 17] 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* setting in PICP.

Complete ICPs with a polynomial-time solvable problem on the lower level are solvable in polynomial time [3]. One can easily adapt the proof in [3] by changing the corresponding constraints to show the following theorem:

Theorem 2.9 *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+) and only weight decrease (ICP-W-), respectively, are also solvable in polynomial time.*

If the lower-level problem is an LP and a unique lower-level solution is fixed on the upper level, the single-level reformulation based on strong duality or KKT conditions simplifies to an LP. Combinatorial methods for ICPs typically rely on problem-specific solution methods for the underlying combinatorial problem or closely related problems. For example, we can reduce the inverse shortest path problem to a shortest path problem, solve the inverse assignment problem via an assignment problem [3], and solve the inverse min cut problem based on a max-flow evaluation [18]. Algorithms for some ICPs with additional integer requirements are given in [19]. For further results on ICPs, we refer the reader to the surveys [20, 21] and references therein.

Complete ICPs with a mixed-integer linear optimization problem on the lower level are co$\mathcal{NP}$-complete [22]. There are solution methods based on enumeration of the lower-level feasible solutions [23, 24].

Partial inverse combinatorial problems (PICPs)

Unlike complete ICPs, which remain polynomial-time solvable whenever the underlying follower problem is, the partial inverse variant (PICP) can be $\mathcal{NP}$-hard even if the lower-level problem itself is 'easy'. Table 1 summarises complexity results for PICPs with some particularly well-known polynomial-time solvable combinatorial problems on the lower level. Moreover, some PICP versions that admit polynomial-time algorithms become $\mathcal{NP}$-hard once we restrict to nonnegative weight changes. Note that hardness results also hold for the case without forbidden elements, i.e. $F = \emptyset$, as most authors only consider this setting

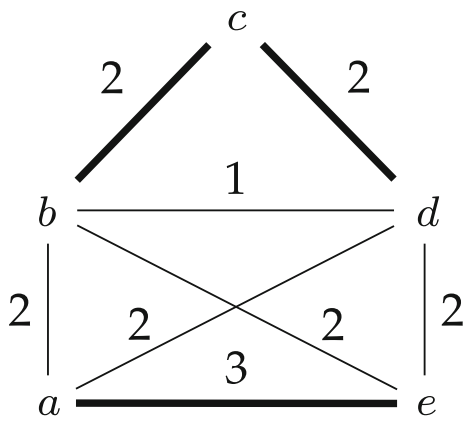

Fig. 4 Example for the partial inverse min spanning tree problem with $R = \{\{a, e\}, \{b, c\}, \{c, d\}\}$, $F = \emptyset$: For the triangle $b - c - d$, increasing the weight of edge $\{b, d\}$ by one is better than decreasing the weights for edges $\{b, c\}$ and $\{c, d\}$ by one each. For the edge $\{a, e\} \in R$ to be in a min spanning tree, decreasing its weight by one is better than increasing the weight of either the two edges $\{a, b\}, \{a, d\}$ or the two edges $\{e, b\}, \{e, d\}$, each by one. In total, the minimal weight modification by either only increasing or decreasing is 3. In contrast, the optimal solution with objective value 2 is to increase the weight of $\{b, d\}$ by one and to decrease the weight of $\{a, e\}$ by one

anyway. To the best of our knowledge, no dedicated solution methods exist for $\mathcal{NP}$-complete PICPs.

If the lower-level problem can be solved as an LP, testing feasibility for a candidate weight-modification vector w for PICP boils down to comparing the optimal objective value of two LPs with modified cost vector $d + w$. In particular, w is feasible if and only if the two LPs

$$\min\left\{(d+w)^\top y : y \in Y\right\} \quad \text{and} \quad \min\left\{(d+w)^\top y : y \in Y,\, y_e = 1\ \forall e \in R,\, y_e = 0\ \forall e \in F\right\}$$

have the same optimal value. As noted in Section 2.4, we can carry out these LP comparisons for PISPP on DAGs, PIAP, and PIMCP by efficient combinatorial methods on suitably reduced graphs.

When discussing PICPs, we have 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 example, Figure 4 shows a partial inverse min spanning tree problem instance where neither exclusively decreasing nor increasing reaches an optimal solution.

Whenever there are no bounds on the weight modifications and no forbidden elements, it suffices to consider only decreases for a number of classical problems, namely shortest paths on DAGs, assignment, and minimum s–t-cuts [3]. Concretely, if for a complete ICP we can assume without loss of generality that all optimal modifications are non-positive, then the corresponding PICP also admits an optimal solution with only decreases. This is shown for the inverse min-cut problem in [4, Lemma 3.1]. The same argument carries over directly to the assignment and shortest-path on DAGs cases via their complete ICPs [3].

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

Several PICPs with no forbidden elements F and only weight decreases (PICP-W-) can be solved in polynomial time by a two-step minimal-completion approach. First, we compute

Table 1 Complexity results for different PICPs with $F = \emptyset$

	W-		W+	
Min Spanning Tree	$\mathcal{P}$	[25]	$\mathcal{NP}$-complete	[26]
Min Spanning Tree with $\|R\| = 1$	$\mathcal{P}$		$\mathcal{P}$	[26]
Min Basis of Matroid	$\mathcal{P}$	[27]	$\mathcal{NP}$-complete	
Shortest path (already feasibility)	$\mathcal{NP}$-complete	Theorem 2.10	$\mathcal{NP}$-complete	
Shortest path on DAG	$\mathcal{P}$	Theorem 2.11	$\mathcal{NP}$-complete	
Shortest path on DAG with $\|R\| = 1$	$\mathcal{P}$		$\mathcal{NP}$-complete	Theorem 3.1
LP in standard form with opt. 0-1 solution	$\mathcal{P}$	[5]	$\mathcal{NP}$-complete	
Assignment	$\mathcal{P}$	[5]	$\mathcal{NP}$-complete	
Assignment with $\|R\| = 1$	$\mathcal{P}$		$\mathcal{NP}$-complete	Theorem 3.2
Min cut	$\mathcal{NP}$-complete	[4]	$\mathcal{NP}$-complete	Theorem 3.3

For problems below the middle horizontal line, results for W- also apply to general weight modifications. Table entries without an explicit reference follow directly from other table entries

a minimal lower-level completion (see Definition 2.3). Second, we solve the corresponding complete ICP for this lower-level completion to find the smallest non-positive weight modifications that make it lower-level-optimal.

This two-stage procedure yields an optimal solution in polynomial time for the minimum spanning tree problem [25], and more generally for the minimum basis problem in a matroid [27]. Here a greedy algorithm yields the minimal completion. Based on the fundamental circuit or fundamental cocircuit of the minimal completion, one can compute the according optimal weight changes.

Moreover, the two-step method based on a minimal lower-level completion solves the PICP optimally if the lower-level problem admits an LP model in standard form, i.e. $\min_y \{d^\top y \mid Ay = b, x \geq 0\}$ with the property that for every cost vector there exists an optimal 0-1 solution. For example, the assignment problem admits an LP in standard form with every optimal basic solution being integral. Hence, PIAP meets our assumptions and can be solved with this method [5]. Be aware, however, that when we rewrite a combinatorial problem as an LP in standard form, we often introduce slack variables that do not appear in the objective. Those slack variables must carry zero weight-modification. Therefore, having a polynomial-time LP model does not automatically make the corresponding PICP easy.

In contrast, PISPP on a general (cyclic) graph is $\mathcal{NP}$-complete. If we simply fix the required arcs and solve the usual flow-LP, we risk getting zero-cost cycles instead of a simple s–t path. In fact, deciding whether there is any feasible weight modification so that an optimal simple path contains a given set of arcs is already $\mathcal{NP}$-hard:

Theorem 2.10 *PISPP on a directed graph is $\mathcal{NP}$-complete.*

Proof For a PISPP instance on a general directed graph to be feasible, every required arc $(a, b) \in R$ that is not incident to a terminal, i.e. $s \neq a$ and $b \neq t$, must lie on some simple s-t-path. In particular, we need vertex-disjoint s-a- and b-t-paths. This is the 2-paths-problem which is $\mathcal{NP}$-complete [28, Lemma 3]. □

In contrast, on a directed acyclic graph (DAG) we can easily determine whether corresponding vertex-disjoint paths exist. In fact, if there are no forbidden arcs and the leader may arbitrarily decrease the weights of the required arcs, then PISPP on a DAG admits a polynomial-time algorithm.

Theorem 2.11 *PISPP on a DAG without forbidden arcs is solvable in polynomial time by only reducing weights of required arcs.*

Proof If $|R| = 1$, first compute a shortest path and a shortest path including R. Then, reduce the weight of the required arc by the weight difference of a shortest path completion and a shortest path with respect to the initial weights. If $|R| > 1$, we solve PISPP by iteratively reducing the weights of the required arcs using a topological ordering. Start with the arc $a_1 \in R$ that is closest to the source. To reduce the weight of a_1, use the tail of the second closest required arc $a_2 \in R$ as preliminary sink. 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. Thus, PISPP on a DAG with arbitrary weight decreases is solvable via a number of shortest path evaluations linear in $|R|$. □

Like PISPP on general graphs, PIMCP is $\mathcal{NP}$-hard. We can formulate the min cut problem as an LP with variables corresponding to the vertices and to the arcs. However, only variables corresponding to arcs appear in the objective function. Thus, we can only modify the capacities of arcs while vertex variables carry zero cost and this remains fixed. Consequently, the LP-based approach described above fails here. In fact, PIMCP is $\mathcal{NP}$-hard, both for arbitrary weight modifications and only weight decreases [4].

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

To the best of our knowledge, the only PICP with only weight increases (PICP-W+) treated in the literature is the partial inverse min spanning tree problem with no forbidden elements. In that setting, the case $|R| = 1$ admits a polynomial-time solution, but the problem is $\mathcal{NP}$-complete without this restriction [26]. In Section 3, we show that the PICP-W+ for both the shortest path problem on a DAG and the assignment problem is $\mathcal{NP}$-hard, even for $|R| = 1$ and $F = \emptyset$. Furthermore, we prove $\mathcal{NP}$-hardness for PIMCP-W+.

Other norms

While we focus on the ℓ_1-norm of the weight modifications, other objective functions are possible, e.g. see [20] and references therein. In particular, PICPs with ℓ_∞ have been studied as preprocessing problems for e.g. shortest path and assignment problem [29].

2.6 Solution approaches

Solution approaches for bilevel problems such as PICPs include singe-level reformulations, decomposition approaches, or branch-and-bound schemes. Next, we briefly review the single-level and decomposition approaches for PICPs. Section 4 then gives details of our branch-and-bound algorithm.

Single-level reformulation: via KKT conditions or strong duality

If the follower's problem is an LP parametric in the leader's variables, we can reformulate the bilevel PICP as a single-level program by replacing the lower level with its primal and dual constraints plus appropriate optimality conditions. There are two standard ways to impose those optimality conditions: Using KKT (complementarity) conditions introduces many local

bilinear constraints. In the single-level reformulation based on strong duality, there is a single – but more complex – bilinear constraint. In both formulations the nonlinearity arises because the lower-level objective is bilinear in the upper-level weight changes and the lower-level decision vector.

In the dual of the lower-level problem, the upper-level variables w appear in the dual constraints rather than in the objective. Introducing dual variables λ, the dual problem is

$$\begin{aligned} \max_{\lambda} \quad & \lambda^\top b \\ \text{s.t.} \quad & \lambda^\top A \leq d + w. \end{aligned}$$

Note that these dual constraints are linear in both the dual lower-level variables λ and upper-level weight modifications w, even though the primal objective is bilinear in w and y.

To enforce optimality of the lower-level solution, we add the complementary-slackness conditions

$$0 = y_e(\lambda^\top A_e - d_e - w_e) \tag{1}$$

for each index e of the primal variables. Together with primal feasibility and dual feasibility, these complementarity constraints are necessary and sufficient for y to be optimal with respect to the modified follower weights $d + w$. Hence, we can rewrite the bilevel PICP as the following single-level nonconvex program:

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

Alternatively, we can impose strong duality instead of complementary slackness by adding the single bilinear constraint

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

When the leader fixes a complete lower-level solution y, both the KKT-based and strong-duality reformulations simplify to LPs.

If only a partial lower-level solution is fixed on the upper level, some bilinear terms remain. In the KKT approach there is a bilinear equation for every lower-level variable. We obtain an LP relaxation if we fix all lower-level variables that have known values and drop those constraints that remain bilinear. We will use a tightened version of this relaxation to generate valid lower bounds, see Section 4.2.2.

Finally, we can convert the full nonlinear reformulation into a MILP via big-M linearisation. However, since the weight-modification variables w can be unbounded, choosing appropriate M is delicate.

Decomposition

A straightforward decomposition approach for PICP is to enumerate the (finite) set of all lower-level-feasible solutions for enforcing follower optimality:

$$
\begin{aligned}
\min_{w,y} \quad & \sum_{e \in E} |w_e| \\
\text{s.t.} \quad & y_e = 1 && (e \in R) \\
& y_e = 0 && (e \in F) \\
& l^w \leq w \leq u^w \\
& Ay = b \\
& y \geq 0 \\
& (d+w)^\top y \leq (d+w)^\top z && \left(z \in \left\{z \in \{0,1\}^n : Az = b\right\}\right)
\end{aligned}
$$

The last set of bilinear inequalities guarantees that no alternative feasible solution z has strictly lower cost than the chosen completion y. A standard way to linearise these is to use big M-constraints. In our implementation, we use indicator constraints provided by the used solver.

Instead of enumerating all exponentially many bilinear constraints for lower-level feasible solutions z, it suffices to only add those that are otherwise violated within a decomposition approach. In order to find constraints that we have to add to the master problem, we 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 completion, the corresponding constraint has to be added to the master problem. Otherwise, i.e. if the objectives are the same, we have found optimal weight modifications.

This decomposition approach is a straightforward generalization of cutting-plane schemes that have been proposed for inverse mixed-integer linear programs [23, 24] or interdiction problems [30].

In this scheme, we can use fast combinatorial methods to solve the lower-level problem. Methods that generate several lower-level optimal solutions at once are especially useful. For example for PISPP, we can enumerate all shortest paths by using Dijkstra's algorithm with only a small overhead compared to evaluating a single shortest path. For PIMCP, we can extract up to two distinct minimum cuts from one max-flow computation by swapping source and sink.

2.7 Pessimistic PICPs

So far and after this subsection, we consider the optimistic version. For PICPs, solving the optimistic or the pessimistic version requires virtually the same algorithm even though there are different objectives on the two levels.

Theorem 2.12 *(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 and no proper superset 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. Suppose to the contrary that $w^* \neq 0$ and there is a pessimistic optimal solution $(\hat{w}, \hat{y})$. This implies that we have $(d + \hat{w})^\top \hat{y} < (d + \hat{w})^\top y$ for all $y \in Y$. These inequalities are strict, so they remain valid under any sufficiently small perturbation of $\hat{w}$. Since $w^* \neq 0$ and a pessimistic optimum cannot improve on the optimistic one, we also have $\hat{w} \neq 0$. We can thus pick a $\bar{w}$ between 0 and $\hat{w}$ so that $\|\hat{w} - \bar{w}\|$ is arbitrarily small but $\hat{w} \neq \bar{w}$. For $\bar{w}$ sufficiently close to $\hat{w}$, the solution $(\bar{w}, \hat{y})$ is still feasible in all three problem cases PICP, PICP-W+ and PICP-W-, but also strictly improves the upper-level objective. This contradicts the assumed optimality of $(\hat{w}, \hat{y})$.

2. Let (w^*, y^*) be an optimistic optimal solution, so we have $(d + w^*)^\top y^* \leq (d + w^*)^\top y$ for $y \in Y$. We now show how to perturb w^* by some $\varepsilon > 0$ so that y^* becomes the unique follower optimum and hence a pessimistic solution while only increasing the upper-level cost by ε. 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. Then for any other $y \neq y^*$, there is at least one coordinate e with $y^*_e = 0$ but $y_e = 1$ and so y^* is now strictly cheaper than y. Thus, (w^p, y^*) is pessimistic bilevel-feasible with upper-level objective value $|w^p| = |w^*| + \varepsilon$. For PICP-W- (and PICP), we can apply a similar construction, decreasing the weight w^* of all elements from $\{e \mid y^*_e = 0\}$ by an equal share of ε. □

The condition in the first part of Theorem 2.12 is easy verifiable, and the condition in the second part becomes equally straightforward once we have a candidate solution y^*. In fact it is necessary for PICP-W+ that no proper subset of y^* is again a feasible lower-level solution. If there were another lower-level solution $\bar{y}$ strictly contained in y^*, then any weight increase that penalizes $\bar{y}$ would also affect y^*, so y^* could never become the unique pessimistic optimum. Similarly, for PICP-W- there must not be any proper superset of y^* that is again a feasible lower-level solution. Additional elements in a lower-level feasible solution $\tilde{y}$ that strictly contains y^* with zero weight could not be prohibited.

In fact, the pessimistic problem can even become infeasible if y^* is bilevel-feasible but $\bar{y}$ or $\tilde{y}$, respectively, is not. Note that the optimistic case can be feasible in these cases with zero-weight elements.

3 Complexity results

3.1 PISPP with only weight increases (PISPP-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 $|R| = 1$ and $F = \emptyset$.*

Our hardness proof is an adaptation of the proof for $\mathcal{NP}$-completeness of the preprocessing shortest path problem on a DAG [29]. Note that PISPP differs from the preprocessing version in its objective. The preprocessing shortest path problem minimizes the maximum single-arc modification, whereas PISPP minimizes the sum of absolute changes. Compared to the reduction from [29], we use different weights for the so-called shortcut arcs, and a slightly simplified graph (omitting intermediate arcs that link the variable and clause gadgets). Combinatorial properties of the original reduction in [29] that do not depend on the precise arc costs remain valid.

Proof We prove $\mathcal{NP}$-completeness by a reduction from 3-SAT. Let an instance of 3-SAT be given by its variables $X = \{x_1, ..., x_n\}$ and clauses $B = \{B_1, ..., B_m\}$ where each clause

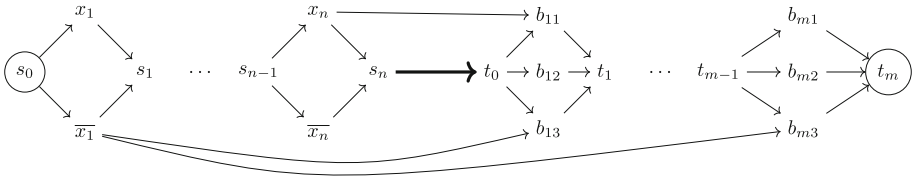

Fig. 5 Sketch of complexity proof graph with $x_1 = b_{13}$, $x_1 = b_{m3}$, $\overline{x_n} = b_{11}$

consists of up to three literals $B_j = \{b_{j1}, b_{j2}, b_{j3}\}$ for $j \in [m]$. In the following, we build 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.

Construction of instance We construct an acyclic digraph consisting of two parts. The first part corresponds to the variables and the second part corresponds to the clauses. Both parts are similarly structured with parallel paths between auxiliary vertices. A path in the first part encodes a truth assignment while a path in the second part encodes which clauses are satisfied by which literal. We connect these two parts by one arc (s_n, t_0) that is the unique element in R and further so-called shortcut arcs enforcing consistency. For a sketch, see Figure 5. We include different costs c per unit weight increase by replacing arcs by the according number of parallel arcs and introducing auxiliary 'inbetween'-vertices.

In the first part, we connect vertices $x_i, \overline{x_i}$ for $i \in [n]$ and $s_0, ..., s_n$ 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 $i \in [n]$. In the second part, 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. We connect vertices $t_0, ..., t_m$ between 'parallel' vertices b_{jk} by arcs $(t_{j-1}, b_{jk}), (b_{jk}, t_j)$ for $j \in [m], k \in [3]$. A t_0-t_m-path corresponds to choosing one literal from every clause. All arcs described so far have unit weights d and costs c are one unit per unit weight increase.

For every literal b_{jk}, we add a shortcut arc 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 $2(n - i + j)$ such that an s_0-s_n-path using a shortcut arc is exactly one unit shorter than a path via (s_n, t_0). Let the cost c for weight increase of a shortcut arc be $n + 2m + 1$.

This construction can be done in polynomial time. In total, we have:

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

By construction, the following observations hold:

1. Vertices $x_i, \overline{x_i}$ have exactly one ingoing arc (s_{i-1}, x_i) and $(s_{i-1}, \overline{x_i})$, 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_m 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 initial weight $d(P) = 2n + 2m + 1$. Every s_0-t_m-path Q using a shortcut arc has initial weight $d(Q) = 2n + 2m$.
4. Let P be an s_0-t_m-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_m-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 respect to weights $d + w$.

Forward-direction Let x^* be a satisfying truth assignment. Since each clause $B_j = \{b_{j1}, b_{j2}, b_{j3}\}$ is satisfied by x^*, we may reorder its literals such that b_{j1} is true under x^* for all $j \in [m]$. Let the set of 'chosen vertices' V^* be the following (sub)set of true literals

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

Then there is a unique s_0-t_m-path P through all vertices in V^* (and all auxiliary vertices $s_i, t_j, i \in [n]_0, j \in [m]_0$) that includes the edge (s_n, t_0). Next, we define the weight-modifications w. 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:

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

Since exactly one variable arc and two clause arcs are penalized for each of the n variables and m clauses, the total weight increase is $n + 2m$. We claim that under the new costs $d + w$, the path P is a shortest s_0-t_m- path and necessarily uses the required arc (s_n, t_0).

Next, we prove that P is a shortest path in G with respect to $d + w$. By construction, P avoids all penalised arcs, thus has the same weight with respect to $d + w$ as with respect to d. Using the third observation, we obtain

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

Due to only non-negative weight modifications, any other s_0-t_m-path without a shortcut arc has modified weight at least as large as P.

Let Q be an s_0-t_m-path that contains a shortcut arc. Then, the initial length of Q is one unit shorter than P, i.e. $d(Q) = 2n + 2m$. 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 the 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 Conversely, suppose there is a weight-modification w with $\sum_a c_a w_a \leq n + 2m$ such that P is a shortest s_0-t_m-path with respect to $d + w$ that includes (s_n, t_0). 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^*. As mentioned in Section 2.5, PISPP-W+ is in $\mathcal{NP}$, which completes the proof. □

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 $|R| = 1$ and $F = \emptyset$.*

Our result for PIAP-W+ strengthens the $\mathcal{NP}$-completeness-result for PIAP with general bounds on the allowed weight modifications [31]. Our proof is a reduction from PISPP-W+ to PIAP-W+ that is based on the analogue result for ℓ_∞ instead of ℓ_1 [29, Theorem 7].

Proof For $\mathcal{NP}$-hardness of PIAP-W+ with $|R| = 1$ and no forbidden edges, we reduce from PISPP-W+ on a DAG with $|R| = 1$ and $F = \emptyset$, see Theorem 3.1. Let an instance for PISPP-W+ on a weighted acyclic digraph $G = (V, A)$ with two distinct vertices s, t, a single required arc a and no forbidden arcs 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 construct an undirected bipartite graph $G' = (V', E)$ whose perfect matchings correspond bijectively to s-t paths in G. For every non-terminal vertex $v \in V \setminus \{s, t\}$, we create two vertices $v_{\text{in}}, v_{\text{out}}$ in V' and add a *node-splitting edge* $\{v_{\text{in}}, v_{\text{out}}\}$ of initial weight 0 to E. Denote the set of all node-splitting edges by E_{NS}. For each directed arc $(u, v) \in A$, we add

an undirected edge between u_{out} and v_{in} of initial weight $d_{\{u,v\}}$ to E. By construction, the obtained undirected graph G' is bipartite and there is a direct correspondence between paths in G and assignments in G': Given an s-t-path in G, the corresponding edges in G' together with the node-splitting edges for all nodes that are not on the path form an assignment in G'. Given an assignment in G', the symmetric difference with E_{NS} is an s-t-path in G' that directly corresponds to an s-t-path in G. The orientation of corresponding arcs in G are correct by construction. Note that the assignment and s-t-path have the same initial weights in both cases. Let the required edges R' contain exactly the edge corresponding to a, and let $F' = \emptyset$.

Without loss of generality, we can assume that weights of edges in E_{NS} cannot be modified by replacing each with sufficiently many parallel edges. We argue that having $|A| + 1$ such parallel edges is sufficient: For the sake of contradiction, suppose (w^*, y^*) is an optimal solution of the PIAP-W+ instance with an additional weight of $\bar{w} > 0$ on all of the $|A| + 1$ copies of some node-splitting edge $\{v_{\text{in}}, v_{\text{out}}\}$. By Lemma 2.5 and optimality of y^*, none of these parallel edges is included in y^*. Thus, edges incident to v that correspond to arcs in G are included in y^*, i.e. the corresponding path in G passes through v. Instead increasing the weights of all (non-node-splitting) edges on alternative paths by $\bar{w}$ each, results in the same path being optimal for the follower. However, there are less than $|A| + 1$ edges in total on alternative paths, contradicting the optimality of (w^*, y^*). Hence, we can assume that no optimal solution modifies the weights of node-splitting edges. Thus, an optimal solution of PIAP-W+ gives us an optimal solution of PISPP-W+ with the same objective value, and vice versa. By the $\mathcal{NP}$-hardness of PISPP-W+ shown in Theorem 3.1, PIAP-W+ is also $\mathcal{NP}$-hard. Moreover, as mentioned in Section 2.5, PIAP-W+ is in $\mathcal{NP}$, which concludes the proof. □

3.3 PIMCP with only weight increases (PIMCP-W+)

Theorem 3.3 *The partial inverse min cut problem with only weight increases (PIMCP-W+) is $\mathcal{NP}$-complete.*

Proof We show that already checking feasibility for PIMCP-W+ is $\mathcal{NP}$-complete by a reduction from the 2-paths-problem [28, Lemma 3], similarly to the proof of Theorem 2.10. Let an instance of the 2-path-problem be given, i.e. for a directed graph G with vertices s_1, t_1, s_2, t_2 we ask whether there exist vertex-disjoint s_1-t_1- and s_2-t_2-paths.

First, we construct a corresponding PIMCP-W+ instance: Let G' be a copy of the graph G where we add the arc (t_1, s_2) if not already present. Let arc capacities be one for all arcs in G'. Then, our PIMCP-W+ instance is to only increase capacities such that there is a minimal cut in G' such that the vertex t_1 is on the same side of the cut as the source s_1 and vertex s_2 is on the same side of the cut as the sink t_2. Clearly this reduction is polynomial. We show that the constructed PIMCP-W+ instance is feasible if and only if the instance of the 2-paths-problem is a yes-instance.

Assume the PIMCP-W+ instance is feasible. Then there exist capacity increases w such that a minium s_1-t_2-cut respects the above same-side constraints. By the max-flow/min-cut theorem this implies that there is a maximum flow from s_1 to t_2 that saturates every arc in the cut, including (t_1, s_2). In particular, there are vertex-disjoint s_1-t_1- and s_2-t_2-paths as the flow would otherwise use backward-crossing arcs and thus would not be maximal. All arcs of vertex-disjoint s_1-t_1- and s_2-t_2-paths in G' are also present in G, so the 2-paths-problem instance is a yes-instance.

For the reverse direction, let there be vertex-disjoint s_1-t_1- and s_2-t_2-paths. Assign all nodes of the s_1-t_1-path to the source side of the cut and all nodes of the s_2-t_2-path to the sink side. In addition, assign all nodes that can be reached from the source without using nodes of the s_2-t_2-path to the source side. Assigning all remaining vertices to the sink side completes the definition of a partition of the vertices. Increase the capacity of arcs that neither cross the cut forward nor backward until all forward crossing-arcs can be saturated. This is a feasible solution of PIMCP-W+. □

Note that the same reduction establishes $\mathcal{NP}$-completeness also for the alternative definitions of PIMCP-W+ mentioned in Section 2.4.

4 Branch and bound for PICP-W+

We solve PICP-W+ (as specified in (PICP)) by a branch-and-bound method with a focus on novel bounding procedures. Throughout, we illustrate our ideas on the three examples of PICP-W+ with shortest path on a DAG, assignment, and min cut, see Section 2.4.

We branch on the upper-level copies of the lower-level variables, which we can assume to be binary. Fixing such a variable to one or zero directly corresponds to adding the according lower-level element to the set of required elements R or forbidden elements F, respectively. Thus, each node in the branch-and-bound tree is a PICP-W+ for enlarged sets $R^{(i)} \supseteq R$ and $F^{(i)} \supseteq F$. We identify a node in the branch-and-bound tree by its sets of required and forbidden elements $R^{(i)}$ and $F^{(i)}$, respectively. Once $R^{(i)}$ and $F^{(i)}$ specify a unique lower-level solution, that node is a leaf. We can solve the resulting complete ICP efficiently by a single-level LP and denote its optimal objective value by $c_{\text{inv}}(R^{(i)}, F^{(i)})$. Based on solving the complete ICP for a minimal lower-level completion, we describe a diving heuristic that provides strong primal bounds in Section 4.1.

Then, in Section 4.2, we introduce two dual-bounding schemes, stemming from a single-level reformulation based on the KKT conditions or strong duality, see Section 2.6. The first dual-bounding scheme, described in Section 4.2.1, is based on the strong duality reformulation. This bound only depends on the minimal completion value $\ell(R^{(i)}, F^{(i)})$. While it provides reasonable bounds already near the top of the branch-and-bound tree, it might not be tight at leaves. As some information can be re-used, it is computationally quite powerful. In order to make best use of the first bounding scheme, we base the node selection on the minimal completion value $\ell(R^{(i)}, F^{(i)})$. The second dual-bounding scheme, described in Section 4.2.2, is based on the KKT conditions and is a stronger version of the first dual-bounding scheme. It also depends on $R^{(i)}$ in addition to $\ell(R^{(i)}, F^{(i)})$ and becomes tighter when approaching leaves. However, as information cannot be re-used, the second dual-bounding scheme is more expensive to evaluate. In our implementation, whenever a node's set of required elements $R^{(i)}$ is a subset of the root's minimal completion used for the primal bound, we only apply the first dual-bounding method.

Afterwards, in Section 4.3, we describe branching rules based on the dual bounds and problem-specific structure. Furthermore, we use propagation to directly add implicitly required or forbidden elements to $R^{(i)}$ and $F^{(i)}$, respectively, and prune subtrees by infeasibility, see Section 4.4. For an overview of the branch-and-bound method, see Algorithm 2.

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

At each node $(R^{(i)}, F^{(i)})$, we get a primal bound by taking the node's minimal lower-level completion and solving the corresponding complete ICP. Recall that the minimal lower-level completion is a lower-level feasible solution y of minimal initial weight $d^T y$ that contains all required elements $R^{(i)}$ but no forbidden elements from $F^{(i)}$, see Section 2.2. In our setting, we determine a minimal lower-level completion by an LP or by problem-specific combinatorial methods, see Section 2.4. Recall that we can solve the corresponding complete ICPs in polynomial time whenever the lower-level problem can be solved in polynomial time [3]. Since the minimal completion corresponds to a leaf of the branch-and-bound (sub-)tree, this strategy is a diving heuristic. In our experiments, it yields quite strong incumbents already at the root node.

4.2 Dual bounding

4.2.1 Required interdiction budget (RIB) value

Starting from the single-level reformulation based on strong duality, we obtain a relaxation of PICP by replacing the dual objective with a lower bound on the follower's objective value in a bilevel-feasible solution. In the following, we show that this relaxation can be solved as a single-level LP. Furthermore, we describe how to formulate this problem as a complete ICP for shortest path or min cut problems on the lower level. Afterwards, we explain how the optimal objective value of the relaxation is globally valid and can be re-used for pruning throughout the algorithm.

Definition 4.1 (Required interdiction budget (RIB)) The *required interdiction budget (RIB) problem* for ℓ is the following bilevel problem

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

We call its optimal objective value the *required interdiction budget value* for ℓ, and denote it by $r(\ell)$.

Theorem 4.2 *The single-level reformulation for the required interdiction budget problem based on strong duality simplifies to an LP. Thus, the RIB value $r(\ell)$ can be obtained by solving a single-level LP.*

Proof In the RIB problem, the upper level depends only on the lower-level objective value but not on the lower-level variables. As the lower level is an LP, the dual problem $\max_\lambda \left\{ \lambda^\top b \mid \lambda^\top A \leq d + w \right\}$ has the same optimal objective value. Note that the dual problem is 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 combine the selection of the dual variables λ with the upper-level choices. In total, we have

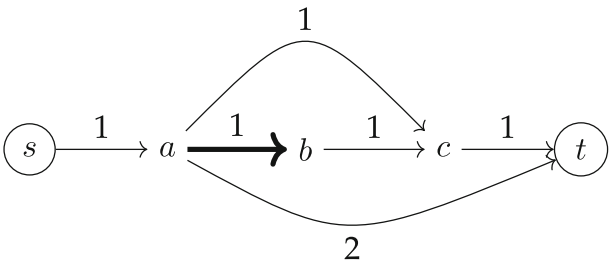

Fig. 6 Example for PISPP with $R = \{(a, b)\}, F = \emptyset$: The RIB value $r(\ell(\{(a, b)\}, \emptyset)) = r(4)$ is one by adding one to the costs of arc (s, a). This is strictly less than the optimal objective value $c^* = 2$ that can be obtained by adding one to the costs of both the arcs (a, c) and (a, t). Thus, the RIB value for the minimal completion length is not a tight bound

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

□

Next, we describe how to reformulate the RIB problem for shortest path and min cut as complete inverse problems on suitably modified graphs. In contrast, it is unclear whether the RIB problem for assignment can be formulated as a complete inverse assignment problem.

Example: shortest path To solve the RIB problem for ℓ for shortest path as a complete inverse shortest path problem, add an additional s-t-path with length ℓ that is disjoint from original non-terminal nodes. Then, require this new additional s-t-path to be optimal on the lower level in the complete inverse shortest path problem.

Example: min cut To solve the RIB problem for ℓ for min cut as a complete inverse min cut problem, add a single arc from the original sink to a new sink with capacity ℓ. This new arc is the only forward-crossing arc of the required cut with all original vertices on one side and the new sink on the other side in the complete inverse min cut problem.

Theorem 4.3 *The RIB value for the minimal completion value $r(\ell(R, F))$ is a lower bound on the PICP's objective value.*

Proof Recall that $\ell(R, F)$ is a lower bound on the lower-level objective value for any bilevel-feasible solution of PICP, see Observation 2.4. If we require the lower-level objective value to be at least ℓ instead of fixing the required elements and forbidden elements, we obtain a relaxation. □

The lower bound $r(\ell(R, F))$ on the PICP's objective value is not tight. For an example with the shortest path problem, see Figure 6.

We can reuse the RIB value $r(\ell)$ by exploiting that it is nondecreasing in the scalar ℓ. Assume that we already pruned some node i due to $r(\ell(R^{(i)}, F^{(i)})) \geq c$, where c is the

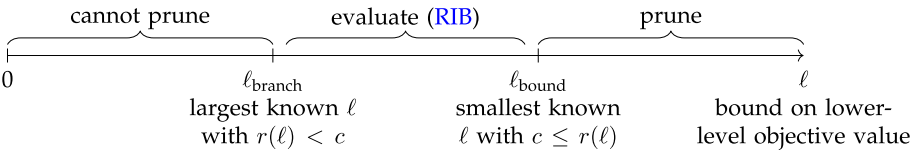

Fig. 7 Strategy on whether to solve (RIB) based on newly encountered $\ell^{(i)}$. The RIB value $r(\ell)$ is nondecreasing in ℓ. If a better incumbent is found, i.e. c is updated, ℓ_{branch} and ℓ_{bound} might decrease

incumbent's objective value. Then, the RIB value of another node i' with at least the same minimal completion length, i.e. $\ell(R^{(i')}, F^{(i')}) \geq \ell(R^{(i)}, F^{(i)})$ will be at least the incumbent's objective value. Thus, we directly prune i' without evaluating the RIB value. Similarly, if a value ℓ did not suffice for pruning, the same holds for all $\ell' \leq \ell$. In order to avoid unnecessarily evaluating the RIB value, we therefore store two values: First, we save ℓ_{bound}, the smallest currently known value for ℓ for which we can prune due to $r(\ell) \geq c$. Second, we save the largest currently known ℓ for which we could not prune due to $r(\ell) < c$ as ℓ_{branch}, see Figure 7 and Algorithm 1. If we find a new incumbent, i.e. the incumbent's objective value c decreases, both ℓ_{bound} and ℓ_{branch} may decrease. While $c \leq r(\ell_{\text{bound}})$ remains valid, this is in general not the case for $r(\ell_{\text{branch}}) < c$. One can simply reset ℓ_{branch} to 0 and keep ℓ_{bound} as it is. However, we store evaluated tuples $(\ell, r(\ell))$ and use them to directly update both ℓ_{bound} and ℓ_{branch}.

Node selection We increase the use of the RIB dual bound by a node selection rule based on the minimal completion value $\ell(R^{(i)}, F^{(i)})$. If ℓ_{bound} is small early on, we can hope to often prune by only a value lookup. This is best achieved by choosing a tree node where the minimal completion value $\ell(R^{(i)}, F^{(i)})$ is minimal.

Algorithm 1 Bound-by-RIB-value-of-min-completions-value(ℓ)

Require: Global variables $\ell_{\text{branch}}, \ell_{\text{bound}}, S$ ▷ Reused for subsequent calls
  if $\ell_{\text{branch}} < \ell < \ell_{\text{bound}}$ **then** ▷ Update ℓ_{bound} or ℓ_{branch}
    Evaluate $r(\ell)$, $S \leftarrow S \cup \{(\ell, r(\ell)\}$ ▷ (RIB)
    if $c \leq r(\ell)$ **then**
      $\ell_{\text{bound}} \leftarrow \ell$
    else
      $\ell_{\text{branch}} \leftarrow \ell$
    end if
  end if
  return $(\ell \geq \ell_{\text{bound}})$ ▷ True or False

4.2.2 Strengthened required interdiction budget (RIB+) value

We can derive a tighter dual bound that explicitly uses the currently required elements. However, this bound is node-specific and cannot be re-used in the same way as the RIB value. Taking the KKT-based single-level reformulation (KKT-1level), we add valid linear cuts and then drop the remaining bilinear constraints. Alternatively, we start with the previous dual bound based on the RIB problem (RIB) and add constraints that are valid in an optimal solution of PICP-W+.

Definition 4.4 (Strengthened required interdiction budget (RIB+)) The *strengthened required interdiction budget (RIB+) problem* for R and F is the following linear problem:

$$
\begin{aligned}
r_+(R,F) = \min_{w,\lambda} \quad & \sum_e w_e \\
\text{s.t.} \quad & \lambda^\top b \geq \ell(R,F) \\
& \lambda^\top A - w \leq d \\
& \lambda^\top A_e - w_e = d_e \qquad (e \in R) \\
& w_e = 0 \qquad (e \in R) \\
& w \geq 0.
\end{aligned}
\tag{RIB+}
$$

The *strengthened required interdiction budget value* $r_+(R,F)$ is the optimal objective value of (RIB+). A *strengthened required interdiction budget solution is an optimal solution* to (RIB+).

Theorem 4.5 *The RIB+ value $r_+(R,F)$ is a valid lower bound for PICP.*

Proof First, we tighten the KKT-based reformulation (KKT-1level) by adding valid cuts. By Lemma 2.5, there are no additional weights for required elements in an optimal solution. Thus, we fix the weight modification for required elements to zero. Furthermore, the lower-level objective value is at least the minimal completion value $\ell(R,F)$ in an optimal solution of PICP-W+, see Observation 2.4. By strong duality, we can use the dual objective $\lambda^\top b$ which only depends linearly on the dual variables λ. Moreover, complementarity constraints corresponding to required or fixed elements simplify to linear constraints. In total, we get the following single-level formulation for PICP-W+:

$$
\begin{aligned}
\min_{w,y,\lambda} \quad & \sum_e w_e \\
\text{s.t.} \quad & y_e = 1 \qquad (e \in R) \\
& y_e = 0 \qquad (e \in F) \\
& Ay = b \\
& \lambda^\top A - w \leq d \\
& w \geq 0 \\
& w_e = 0 \qquad (e \in R) \\
& \lambda^\top b \geq \ell(R,F) \\
& \lambda^\top A_e - w_e = d_e \qquad (e \in R) \\
& y_e(\lambda^\top A_e - d_e - w_e) = 0 \qquad (e \in E \setminus (R \cup F))
\end{aligned}
$$

In order to obtain a linear relaxation, we simply drop the remaining bilinear complementarity constraints for elements that are neither required nor forbidden. In addition, we remove the lower-level constraints for a lower-level completion, since they no longer affect the objective. The resulting problem (RIB+) is therefore a linear relaxation of PICP-W+. □

Alternatively, we can view this stronger bound as a tightened version of the RIB value. Taking the LP formulation for the RIB value from the proof of Theorem 4.2, we add two families of valid cuts: First, by Lemma 2.5, weights of required elements in R are not increased in an optimal solution of PICP-W+. Thus, we forbid weight increases for required elements.

Furthermore, due to complementarity in an optimal solution, dual constraints corresponding to required elements R have to be fulfilled with equality. Note that we cannot add analogous constraints for the forbidden elements F.

By construction, this second dual bound is stronger than the RIB value, i.e. $r(\ell(R, F)) \leq r_+(R, F)$ for all $R, F \subseteq E$. However, the stronger bound $r_+(R, F)$ is only valid for the subtree and cannot be reused elsewhere, unlike the RIB value $r(\ell)$.

Whenever R and F fix a unique lower-level solution, problem (RIB+) is a complete ICP. Thus, this bound is tight at leaves of the branch-and-bound tree.

Example: min cut Recall that in PIMCP additional capacity can be assigned to arcs while nodes are fixed on either side of the cut via R and F. Thus, in order for the RIB+ value to be stronger than the RIB value, we need to include the arcs that are implicitly required to forward-cross the cut. Note that the constraints of the dual of the follower problem are included in the RIB+ problem, i.e. the RIB+ problem for PIMCP contains a flow problem. In the RIB problem, the total flow leaving the source is at least the minimal completion value. In an optimal solution of the max flow problem, arcs that forward-cross the cut are saturated while there is no flow on backward-crossing arcs. Furthermore, in an optimal solution of PIMCP, there is no additional capacity on forward-crossing or backward-crossing arcs. Thus, we can fix the flow of forward- and backward-crossing arcs to the initial capacity and zero, respectively.

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

$T \leftarrow \{R\}$, $\ell_{\text{branch}} \leftarrow 0$, $\ell_{\text{bound}} \leftarrow \infty$, $S \leftarrow \emptyset$ ▷ Global variables
$c \leftarrow c_{\text{inv}}(\arg\min \left\{ d^\top y : y \in Y,\ y_e = 1\ \forall e \in R,\ y_e = 0\ \forall e \in F \right\})$ ▷ Inverse problem for min completion
while $T \neq \emptyset$ **do**
  choose $(R^{(i)}, F^{(i)}) \in T$, remove $(R^{(i)}, F^{(i)})$ from T
  if Bound-by-RIB-value-of-min-completions-value($\ell(R^{(i)}, F^{(i)})$) **then** ▷ Algorithm 1
    continue
  else if $R^{(i)}$ is complete lower-level solution **then**
    if $c_{\text{inv}}(R^{(i)}, F^{(i)}) < c$ **then** ▷ Inverse Problem
      $c \leftarrow c_{\text{inv}}(R^{(i)}, F^{(i)})$
      $\ell_{\text{branch}} \leftarrow \max(\{0\} \cup \{\ell \mid \exists(\ell_{\text{eval}}, r_{\text{eval}}) \in S, r_{\text{eval}} < c, \ell_{\text{eval}} = \ell\})$
      $\ell_{\text{bound}} \leftarrow \min(\{\ell_{\text{bound}}\} \cup \{\ell \mid \exists(\ell_{\text{eval}}, r_{\text{eval}}) \in S, r_{\text{eval}} \geq c, \ell_{\text{eval}} = \ell\})$
    end if
  else if $(R^{(i)}, F^{(i)})$ is subsolution of initial solution (min-completion) **then**
    branch on lower-level variables, add new children to T
  else if $r_+(R^{(i)}, F^{(i)}) < c$ **then** ▷ (RIB+)
    branch on lower-level variables, add new children to T
  end if
end while
return c

4.3 Branching

During branching, we add elements to R and F based on the current minimal completion and solution of the RIB+ problem in the hope to quickly increase our dual bounds. Both dual bounds, i.e. the RIB value and the tighter RIB+ value, are nondecreasing in the minimal completion value $\ell(R, F)$. The minimal completion value might increase whenever we forbid an element that belongs to the previous minimal completion, or require an element that was

not in the previous minimal completion. Furthermore, in the RIB+ problem, there is no weight increase for required elements. Thus, elements with positive weight increase in the current RIB+ solution are interesting candidates for branching.

Since the RIB+ problem depends explicitly on the required elements R, we branch such that we add at least one new element to the required elements R in every child node. Concretely, we identify problem-specific subsets of lower-level elements such that any feasible lower-level solution must include exactly one of them. This also reduces the size of the branch-and-bound tree versus simply branching on whether to add a lower-level element to R or F. Below, we describe problem-specific subsets of lower-level elements for our three examples shortest path, assignment and min cut.

Our choice on which problem-specific subset to branch on is based on the current minimal completion and the current RIB+ solution if available. If there is an element in the minimal completion with additional weight in the RIB+ solution, we branch on a problem-specific set of elements that includes such an element. In case no such element exists, we use a problem-specific set of elements that includes an element with maximal additional weight. If no element got additional weight in the current RIB+ solution or if we skipped solving the RIB+ problem, we choose a random set of problem-specific elements. Although there may be multiple minimal completions or RIB+ solutions, we only use the specific completion and RIB+ solution that we have already computed.

Example: shortest path Any feasible path contains exactly one of the arcs leaving the source. The same is true for arcs entering the sink, or leaving/entering the head/tail of already required subpaths. We choose a vertex where the sum of the additional weights on the out-/ingoing arcs in the current RIB+ solution is maximal. Alternatively, we take a vertex that has maximal out-/indegree.

Example: assignment For every vertex, exactly one incident edge is selected in an assignment. First, we choose an edge in the current minimal completion that has maximal additional weight in the current RIB+ solution. If no such edge exists, we take an edge with maximal additional weight in the current RIB+ solution. If this is also not applicable, we pick a random unfixed edge. Then, secondly, we choose the end vertex of the chosen edge where the sum of all additional weight in the RIB+ solution on incident edges is larger. Alternatively, we take a vertex of maximal degree. Then, in every child node we add exactly one of the unfixed edges incident to the chosen vertex to the set of required elements.

Example: min cut We branch on vertices as vertices are fixed in PIMCP, see Section 2.4. First, if possible, we choose an arc in the current minimal completion with maximal additional weight in the current RIB+ solution. If both end vertices are not fixed yet, we branch at both vertices simultaneously, creating four child nodes for the four combinations of fixing the two end vertices to either side. Otherwise, we branch on an unfixed vertex that is incident to an arc with maximal additional weight in the RIB+ solution. If there is only additional weight on arcs that are already fixed to not cross the cut in the RIB+ solution or the RIB+ solution has not been evaluated, we choose a vertex with maximal degree that is not fixed yet.

4.4 Propagation and pruning by infeasibility

In order to reduce the size of the branch-and-bound tree, we use elementary propagation techniques and try to detect infeasible subtrees as early as possible. Some partial solutions

(R, F) immediately imply additional elements that must appear in every lower-level completion. Whenever we detect such 'forced' elements, we add them to the required elements R. For PISPP and PIAP, vertices of in-/outdegree or degree one are candidates for propagation.

Moreover, if a partial solution (R, F) admits no completion, then its entire subtree is infeasible and can be pruned. If there is a lower-level completion for an instance of PISPP-W+ of PIAP-W+, the instance is feasible by Theorem 2.2.

Example: min cut In a feasible PIMCP-W+ instance, for every forward-crossing arc (v, w), the vertex v must be reachable from the source s within the vertices assigned to the source side S. Thus, if there is a single incoming arc (u, v) with u free, then u must also be placed in S. Analogously, the sink t needs to be reachable from vertices in the complement $\bar{S}$ within $\bar{S}$.

5 Computational Results

5.1 Implementation, software and computing infrastructure

We implemented the following methods in Python 3.8.10, using Gurobi 10.0.2 via the gurobipy interface for all (I)LPs, and NetworkX 3.1 for graph data structures and combinatorial subroutines. Experiments were run on an AMD EPYC 7742 64-core CPU. As our prototype implementation is not parallelized, we restricted each run to a single core and limited Gurobi to one thread to ensure a fair comparison.

Branch and bound We implemented the basic branch-and-bound scheme described in Section 4. For finding shortest path completions, we used the bidirectional Dijkstra algorithm implemented in NetworkX. For underlying assignment or min cut problems, we solved the according LP for minimal completion with Gurobi. We furthermore used Gurobi to solve all LPs for the dual bounds. When computing the RIB value, we actually used a slightly strengthened version with the element-specific constraints from the RIB+ value for the initial required elements of the problem instance. For each of our three example classes, we implemented two different branching rules. The first branching strategy picks an element from a minimal completion with additional weight in the RIB+ solution as described in Section 4.3. The second branching strategy is independent of the dual bounds. Note that extracting variable values from a gurobipy model can incur non-negligible overhead in our prototype implementation.

Single-level reformulation with indicator constraints We implemented the single-level reformulations mentioned in Section 2.6 using Gurobi's built-in indicator constraints. In preliminary experiments, the reformulation based on strong duality solved significantly faster than the one based on complementarity constraints. Thus, we only considered the version via duality for our tests.

Decomposition We implemented the decomposition approach described in Section 2.6 using Gurobi for the master problem and the subproblems for assignment and min cut. For the shortest path problem, we used an implementation of Dijkstra's algorithm from NetworkX. In preliminary tests, the decomposition approach performed poorly in comparison to the other two solution approaches when only a single lower-level solution is added per iteration. For shortest path, we can simply find all currently shortest paths by Dijkstra's algorithm.

Adding them all in a single batch significantly improved runtime. Hence, our computational comparison includes the decomposition approach only for shortest path.

5.2 Instances

We randomly generated instances that satisfy the assumptions, using the number of vertices and edges as input parameters. It turns out that performance of the methods depends to a large degree on the density of the graph for PISPP-W+, PIAP-W+ and PIMCP-W+. Accordingly, we report results on test sets covering a range of densities. Instances are available on request.

Shortest path For PISPP-W+, we randomly generated directed acyclic graphs that tend to have many s-t-paths of roughly the same length. We started with a path on 100 nodes with weights drawn uniformly at random from $\{1, 2, \ldots, 5\}$. Then, we added shortcut arcs until the target number of arcs is reached by choosing two distinct vertices uniformly at random that are not yet adjacent. The weight of a shortcut arc is the number of vertices between its end vertices on the original path plus a random offset drawn uniformly from $\{-5, -4, \ldots, 4, 5\}$. Source and sink for the problem are the terminal vertices of the initial path. A randomly chosen arc from the graph was fixed as the unique required arc in R. Initially, there was no explicitly forbidden arc, i.e. $F = \emptyset$.

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

Assignment For PIAP, we generated random bipartite graphs with a specified number of vertices and edges. First, we split the vertices evenly into two parts. Then, we repeatedly picked one vertex from each of the parts uniformly at random and added an edge between the two vertices, if it was not already present, until the desired number of edges was reached. We assigned each edge a weight drawn uniformly at random from $\{1, 2, \ldots, 10\}$. We either fixed the size of the required edges R in advance or chose the size uniformly at random from the specified set. Then, we picked the according number of distinct edges uniformly at random from all edges in the graph. Initially, there were no explicitly forbidden edges.

Min cut For PIMCP-W+, we generated random directed graphs with the specified number of vertices and arcs as follows: First, from the set of vertices, we picked two distinct vertices to be the source s and sink t. Then, the specified number of arcs are inserted one by one. For each arc, we randomly chose a tail from all vertices except the sink and a head from all vertices except the source. If this would create a self-loop or duplicate an existing arc, we retried. When adding an arc to our graph, we assigned it a capacity drawn uniformly at random from $\{1, 2, \ldots, 10\}$. If the terminal vertices ended up in different connected components, we declared the instance infeasible and discarded it. Otherwise, we deleted any connected components not containing s and t. Finally, we chose a random number of vertices to be randomly fixed on either side of the cut.

5.3 Computations

For all three problems, we ran each method on every instance with a one-hour time limit.

Table 2 Statistics on running times (in seconds) for PISPP-W+, cf. Figure 8.

Number of arcs	Method	Solved	Mean [s]	Std [s]	Min [s]	25% [s]	50% [s]	75% [s]	Max of solved [s]
2 000	B&B: max degree	100	120.4	392.8	0.0	0.5	2.2	26.8	2 396.7
	B&B: RIB+ weights	99	118.9	409.3	0.0	0.8	8.3	54.0	1 190.8
	Gurobi	98	204.8	583.1	0.2	1.3	9.8	122.0	2 379.3
	Decomposition	27	2726.6	1489.0	0.1	2 092.0	3 600.0	3 600.0	2 560.6
3 000	B&B: max degree	88	655.9	1 235.8	0.1	4.4	18.4	384.8	3 362.3
	B&B: RIB+ weights	80	1 138.7	1 467.4	0.1	10.8	134.9	2 511.8	3 561.7
	Gurobi	67	1 511.3	1 607.7	0.3	31.6	588.4	3 600.0	3 191.8
	Decomposition	15	3 114.3	1 194.7	0.2	3 600.0	3 600.0	3 600.0	2 618.0
4 000	B&B: max degree	74	1 059.4	1 549.3	0.2	4.8	63.2	3 600.0	1 819.9
	B&B: RIB+ weights	61	1 630.1	1 670.9	0.2	12.1	719.2	3 600.0	3 459.6
	Gurobi	46	2 173.7	1 660.1	0.4	66.9	3 600.0	3 600.0	3 598.2
	Decomposition	15	3 099.4	1 236.0	0.3	3 600.0	3 600.0	3 600.0	3 166.9

For the calculation of mean, standard deviation, and quartiles, the time limit of 3 600 seconds has been used for instances that have not been solved

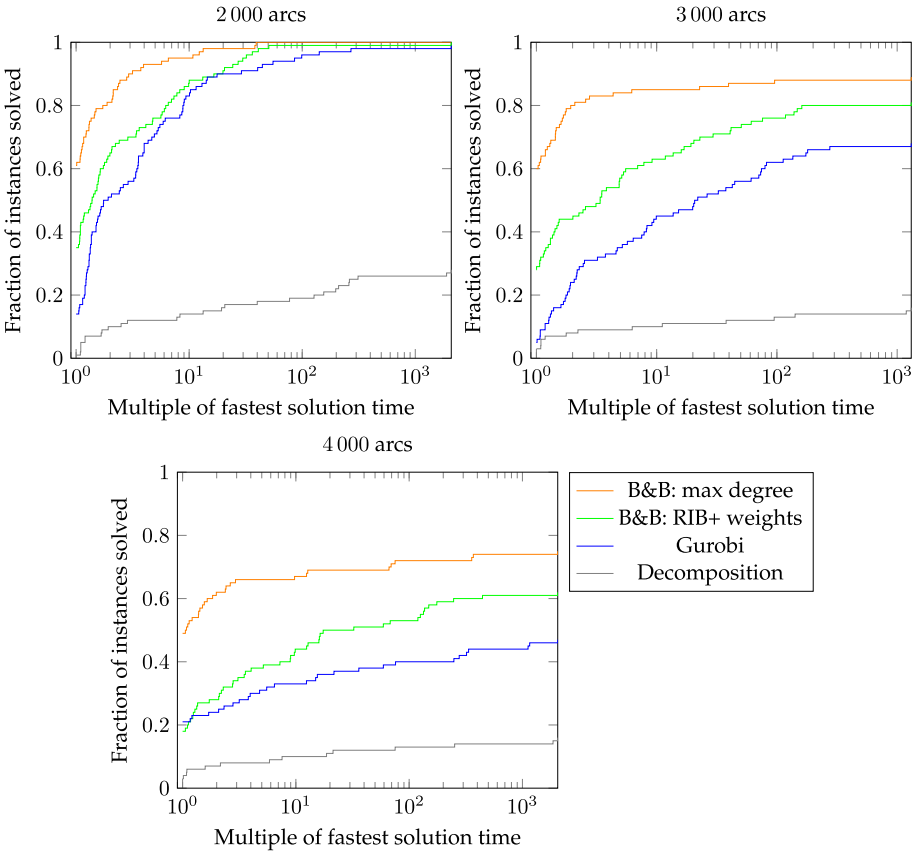

Fig. 8 Performance for PISPP-W+ instances with one required arc and graphs with 100 vertices and 100 instances each. The time limit is 1 hour per instance. For statistics on absolute runtimes, see Table 2

5.3.1 Partial inverse shortest path

We tested all methods on DAGs with 100 vertices and various arc counts, see Figure 8 and Table 2. Denser instances seem to be generally more difficult. Despite their simple implementation, both of our branch-and-bound versions outperformed Gurobi on the single-level duality reformulation and the cutting-plane (decomposition) approach. In particular, our branch-and-bound versions solved significantly more instances to optimality within the one-hour time limit. Interestingly, the simpler branching rule extending the required subpath at a vertex of maximal degree ('B&B: max degree') outperformed the rule based on RIB+ weights ('B&B: RIB+ weights'). A relevant factor might be the non-negligible overhead when retrieving the RIB+ solution from the gurobipy model in contrast to the time needed to determine the degree of a vertex.

The decomposition method, even when enhanced to add all current shortest paths at once, lagged behind both branch-and-bound versions and the single-level reformulation. Note that we chose the instances such that there are many such solutions of same weight.

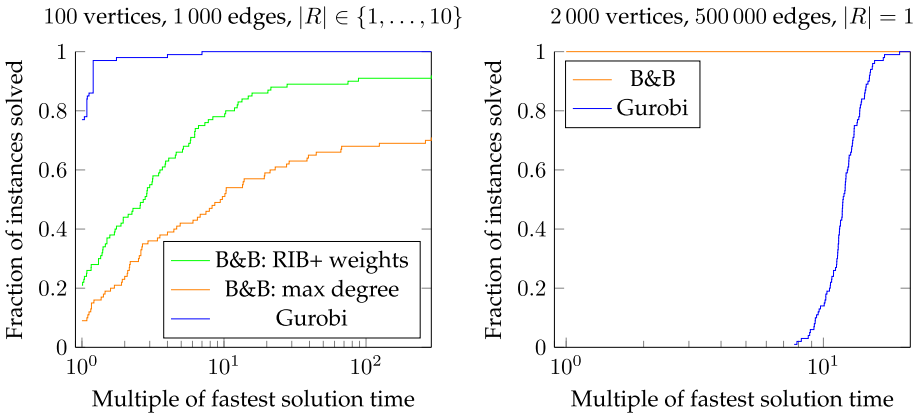

Fig. 9 Performance for PIAP-W+ instances with two different parameter configurations and 100 instances each. The time limit is 1 hour per instance. For statistics on absolute runtimes, see Table 3

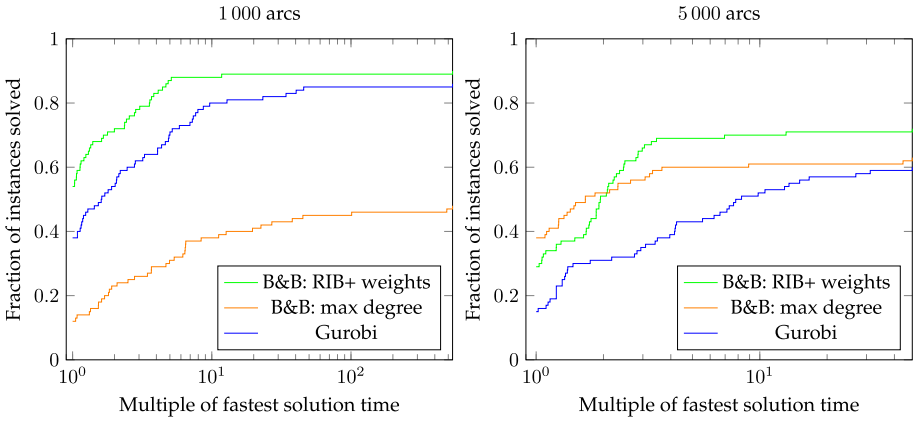

Fig. 10 PIMCP-W+: 100 feasible instances, each with 100 vertices. The time limit is 1 hour per instance. For statistics on absolute runtimes, see Table 4

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. The result is quite lopsided either way, see Figure 9 and Table 3. On dense graphs, our dive heuristic finds an optimal solution and the RIB+ bound at the root exactly matches it. Consequently, no branching is needed. In this case, our method is faster than Gurobi solving the single-level reformulation. Interestingly, Gurobi also solves the single-level reformulation for all these instances at the root node. Note that branching rules are irrelevant in this case, so only one version of our method is shown. For smaller but sparser graphs, Gurobi solving the single-level reformulation outperforms our method, especially if more than one edge is required. In this case, a significant amount of branching is performed by all methods.

Table 3 Statistics on running times (in seconds) for PIAP-W+, cf. Figure 9.

Instances (vertices / edges / $\|R\|$)	Method	Solved	Mean [s]	Std [s]	Min [s]	25% [s]	50% [s]	75% [s]	Max of solved [s]
100 / 1 000 / $\in \{1, \ldots, 10\}$	B&B: RIB+ weights	91	427.5	1 029.1	0.0	0.8	10.8	171.3	1 184.2
	B&B: max degree	70	1 149.1	1 623.5	0.0	1.6	18.4	3 600.0	1 282.7
	Gurobi	100	83.2	222.2	0.2	0.3	2.5	39.4	1 221.9
2 000 / 500 000 / 1	B&B	100	46.9	7.4	32.4	41.6	46.5	50.8	66.3
	Gurobi	100	575.9	100.1	346.1	492.5	574.3	653.1	816.9

For the calculation of mean, standard deviation, and quartiles, the time limit of 3 600 seconds has been used for instances that have not been solved

Table 4 Statistics on running times (in seconds) for PIMCP-W+, cf. Figure 10.

Number of arcs	Method	Solved	Mean [s]	Std [s]	Min [s]	25% [s]	50% [s]	75% [s]	Max of solved [s]
1 000	B&B: RIB+ weights	89	656.1	1 152.1	0.1	7.2	70.3	560.1	2 668.3
	B&B: max degree	47	2 039.3	1 709.2	0.1	14.5	3 600.0	3 600.0	2 490.6
	Gurobi	85	853.5	1 320.9	0.1	16.7	110.7	1 032.2	3 203.8
5 000	B&B: RIB+ weights	71	1 383.9	1 561.3	0.3	25.6	510.6	3 600.0	2 714.9
	B&B: max degree	62	1 571.8	1 671.3	0.3	26.2	499.8	3 600.0	3 272.0
	Gurobi	59	1 821.0	1 643.7	0.5	45.5	1 801.0	3 600.0	3 255.4

For the calculation of mean, standard deviation, and quartiles, the time limit of 3 600 seconds has been used for instances that have not been solved

Table 5 Statistics on running times (in seconds) for 14 infeasible PIMCP-W+ instances with 100 vertices and 1 000 arcs.

Method	Detected	Mean [s]	Std [s]	Min [s]	25% [s]	50% [s]	75% [s]	Max of detected [s]
B&B: RIB+ weights	13	257.4	962.1	0.0	0.1	0.3	0.5	0.9
Gurobi	4	2 572.9	1 685.3	0.1	909.7	3 600.0	3 600.0	12.9

For the calculation of mean, standard deviation, and quartiles, the time limit of 3 600 seconds has been used for instances that have not been solved

5.3.3 Partial inverse min cut

For PIMCP-W+, our branch-and-bound implementation with branching based on additional weight in a RIB+ solution outperforms Gurobi, see Figure 10. For sparse graphs with only 1 000 arcs and 100 vertices, Gurobi outperforms our branch-and-bound method with branching based only on vertex degrees. However, for more dense graphs with 5 000 arcs and 100 vertices, both versions of our branch-and-bound method outperform Gurobi. Here, branching based on the vertex degree is the fastest method for slightly more instances. For more difficult instances, branching based on the additional weight in a RIB+ solution seems preferable as

it allows solving more instances within the one-hour time limit. Recall that retrieving the solution of the RIB+ problem costs significant time in our implementation.

Figure 10 and Table 4 report times only for feasible instances. For detection of infeasible instances as they occurred during the generation of the test set for 1 000 arcs, our branch-and-bound method clearly outperforms Gurobi, see Table 5. Our branch-and-bound method detected infeasibility for 13 out of 14 instances in under one second, whereas Gurobi detected infeasibility only for 4 instances within one hour. For the lone remaining instance, our branch-and-bound method took 6 141.4 seconds to certify infeasibility while Gurobi did not terminate on it even after three hours.

6 Conclusion

This paper proposes one of the first algorithmic contributions to solve $\mathcal{NP}$-complete PICPs in which weights can only be increased. Our algorithm uses a branch-and-bound scheme that pairs a simple yet powerful diving heuristic for primal bounds with two novel dual-bounding schemes exploiting the fact that complete ICPs remain polynomial-time solvable. Our computational results suggest that for the three example test cases using shortest path, assignment and minimum cut problems on the lower level, our prototype implementation is very much competitive with a state-of-the-art solver applied to a single-level reformulation, even though it is implemented in Python and does not enjoy access to all the features of a mature solver. For PISPP-W+ and PIMCP-W+, at least one of our branching strategies allowed us to solve strictly more instances within one hour than Gurobi did for the single-level reformulation. For PIAP-W+, except for very dense or infeasible instances, Gurobi with the single-level reformulation was clearly superior. We observed that density of the graphs is a key factor that generally tends to favour our methods across all three problem types. It is not unusual for our method to solve an instance immediately at the root or after only a few nodes, giving us large speedups. Although using information from the RIB+ solution for branching seems to be beneficial, we believe there is still considerable room for improvement. This suggests that a combination of methods might yield a significantly more powerful method, e.g. by integrating our bounding subproblems into a mature 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.

We also proved $\mathcal{NP}$-completeness for three PICP-W+ variants. However, for such problems one can ask for milder conditions than completeness of the solution provided by R and F on when the problem becomes solvable in polynomial time. Such insights could guide branching decisions. For example, is PISPP-W+ still $\mathcal{NP}$-complete if $F = \emptyset$ and R is a path connected to the source or sink? If the answer were no, we could tailor our search to steer R into that 'easy' configuration as early as possible. Finally, one could try to extend our algorithmic ideas to PICPs with more difficult combinatorial problems on the lower level.

Acknowledgements We thank the two anonymous reviewers for their helpful comments.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability The datasets generated during and analysed during the current study are available from the corresponding author on reasonable request.

Declarations

Competing Interests The authors have no relevant financial or non-financial interests to disclose.

References

1. Dial, R.B.: Network-optimized road pricing: Part I: A parable and a model. Oper. Res. **47**(1), 54–64 (1999). https://doi.org/10.1287/opre.47.1.54
2. Moser, T.J.: Shortest path calculation of seismic rays. Geophysics **56**(1), 59–67 (1991). https://doi.org/10.1190/1.1442958
3. Ahuja, R.K., Orlin, J.B.: Inverse optimization. Oper. Res. **49**(5), 771–783 (2001). https://doi.org/10.1287/opre.49.5.771.10607
4. Gassner, E.: The partial inverse minimum cut problem with L_1-norm is strongly NP-hard. RAIRO - Operations Research **44**(3), 241–249 (2010). https://doi.org/10.1051/ro/2010017
5. Yang, X., Zhang, J.: Partial inverse assignment problems under l_1 norm. Oper. Res. Lett. **35**(1), 23–28 (2007). https://doi.org/10.1016/j.orl.2005.12.003
6. Korte, B., Vygen, J.: Combinatorial Optimization, 6th edn. Springer, Berlin (2018)
7. Kleinert, T., Labbé, M., Ljubic, I., Schmidt, M.: A survey on mixed-integer programming techniques in bilevel optimization. EURO Journal on Computational Optimization 9, 100007 (2021) https://doi.org/10.1016/j.ejco.2021.100007
8. Mersha, A.G., Dempe, S.: Linear bilevel programming with upper level constraints depending on the lower level solution. Appl. Math. Comput. **180**(1), 247–254 (2006). https://doi.org/10.1016/j.amc.2005.11.134
9. Calvete, H.I., Galé, C.: In: Dempe, S., Zemkoho, A. (eds.) Algorithms for Linear Bilevel Optimization, pp. 293–312. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-52119-6_10
10. Fischetti, M., Ljubic, I., Monaci, M., Sinnl, M.: A new general-purpose algorithm for mixed-integer bilevel linear programs. Oper. Res. **65**(6), 1615–1637 (2017). https://doi.org/10.1287/opre.2017.1650
11. Lim, C., Smith, J.C.: Algorithms for discrete and continuous multicommodity flow network interdiction problems. IIE Trans. **39**(1), 15–26 (2007). https://doi.org/10.1080/07408170600729192
12. Israeli, E., Wood, R.K.: Shortest-path network interdiction. Networks **40**(2), 97–111 (2002). https://doi.org/10.1002/net.10039
13. Fischetti, M., Ljubic, I., Monaci, M., Sinnl, M.: Interdiction games and monotonicity, with application to knapsack problems. INFORMS J. Comput. **31**(2), 390–410 (2019). https://doi.org/10.1287/ijoc.2018.0831
14. Furini, F., Ljubic, I., Martin, S., Segundo, P.S.: The maximum clique interdiction problem. Eur. J. Oper. Res. **277**(1), 112–127 (2019). https://doi.org/10.1016/j.ejor.2019.02.028
15. Wei, N., Walteros, J.L.: Integer programming methods for solving binary interdiction games. Eur. J. Oper. Res. **302**(2), 456–469 (2022). https://doi.org/10.1016/j.ejor.2022.01.009
16. Zenklusen, R.: Matching interdiction. Discret. Appl. Math. **158**(15), 1676–1690 (2010). https://doi.org/10.1016/j.dam.2010.06.006
17. Smith, J.C., Song, Y.: A survey of network interdiction models and algorithms. Eur. J. Oper. Res. **283**(3), 797–811 (2020). https://doi.org/10.1016/j.ejor.2019.06.024
18. Ahuja, R.K., Orlin, J.B.: Combinatorial algorithms for inverse network flow problems. Networks **40**(4), 181–187 (2002). https://doi.org/10.1002/net.10048
19. Ahmadian, S., Bhaskar, U., Sanità, L., Swamy, C.: Algorithms for inverse optimization problems. In: Azar, Y., Bast, H., Herman, G. (eds.) 26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland. LIPIcs, vol. 112, pp. 1–1114. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2018). https://doi.org/10.4230/LIPIcs.ESA.2018.1

20. Heuberger, C.: Inverse combinatorial optimization: A survey on problems, methods, and results. J. Comb. Optim. **8**(3), 329–361 (2004). https://doi.org/10.1023/B:JOCO.0000038914.26975.9b
21. Demange, M., Monnot, J.: In: Paschos, V.T. (ed.) An Introduction to Inverse Combinatorial Problems, pp. 547–586. John Wiley & Sons, Ltd, Hoboken/London (2014). Chap. 17. https://doi.org/10.1002/9781119005353.ch17
22. Bulut, A., Ralphs, T.K.: On the complexity of inverse mixed integer linear optimization. SIAM J. Optim. **31**(4), 3014–3043 (2021). https://doi.org/10.1137/20M1377369
23. Wang, L.: Cutting plane algorithms for the inverse mixed integer linear programming problem. Oper. Res. Lett. **37**(2), 114–116 (2009). https://doi.org/10.1016/J.ORL.2008.12.001
24. Bodur, M., Chan, T.C.Y., Zhu, I.Y.: Inverse mixed integer optimization: Polyhedral insights and trust region methods. INFORMS J. Comput. **34**(3), 1471–1488 (2022). https://doi.org/10.1287/IJOC.2021.1138
25. Cai, M., Duin, C.W., Yang, X., Zhang, J.: The partial inverse minimum spanning tree problem when weight increase is forbidden. Eur. J. Oper. Res. **188**(2), 348–353 (2008). https://doi.org/10.1016/j.ejor.2007.04.031
26. Li, X., Zhang, Z., Du, D.: Partial inverse maximum spanning tree in which weight can only be decreased under l_p-norm. J. Global Optim. **70**(3), 677–685 (2018). https://doi.org/10.1007/s10898-017-0554-5
27. Zhang, Z., Li, S., Lai, H., Du, D.: Algorithms for the partial inverse matroid problem in which weights can only be increased. J. Global Optim. **65**(4), 801–811 (2016). https://doi.org/10.1007/s10898-016-0412-x
28. Fortune, S., Hopcroft, J.E., Wyllie, J.: The directed subgraph homeomorphism problem. Theoretical Computer Science 10, 111–121 (1980) https://doi.org/10.1016/0304-3975(80)90009-2
29. Lai, T.-C., Orlin, J.: The complexity of preprocessing. Technical report, Sloan School of Mangement (November (2003)
30. DeNegre, S.: Interdiction and discrete bilevel linear programming. PhD thesis, Lehigh University (2011)
31. Yang, X.: Complexity of partial inverse assignment problem and partial inverse cut problem. RAIRO - Operations Research **35**(1), 117–126 (2001). https://doi.org/10.1051/ro:2001106