An Investigation of the Recoverable Robust Assignment Problem

Dennis Fischer
Department of Computer Science, RWTH Aachen, Germany

Tim A. Hartmann
Department of Computer Science, RWTH Aachen, Germany

Stefan Lendl
Institute of Operations und Information Systems, Universität Graz, Austria

Gerhard J. Woeginger
Department of Computer Science, RWTH Aachen, Germany

Abstract

We investigate the so-called recoverable robust assignment problem on complete bipartite graphs, a mainstream problem in robust optimization: For two given linear cost functions c_1 and c_2 on the edges and a given integer k, the goal is to find two perfect matchings M_1 and M_2 that minimize the objective value $c_1(M_1) + c_2(M_2)$, subject to the constraint that M_1 and M_2 have at least k edges in common.

We derive a variety of results on this problem. First, we show that the problem is W[1]-hard with respect to parameter k, and also with respect to the complementary parameter $k' = n/2 - k$. This hardness result holds even in the highly restricted special case where both cost functions c_1 and c_2 only take the values 0 and 1. (On the other hand, containment of the problem in XP is straightforward to see.) Next, as a positive result we construct a polynomial time algorithm for the special case where one cost function is Monge, whereas the other one is Anti-Monge. Finally, we study the variant where matching M_1 is frozen, and where the optimization goal is to compute the best corresponding matching M_2. This problem variant is known to be contained in the randomized parallel complexity class RNC[21], and we show that it is at least as hard as the infamous problem Exact Red-Blue Matching in Bipartite Graphs whose computational complexity is a long-standing open problem.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Fixed parameter tractability

Keywords and phrases assignment problem, matchings, exact matching, robust optimization, fixed paramter tractablity, RNC

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.19

Funding Stefan Lendl acknowledges the support of the Austrian Science Fund (FWF): W1230. Dennis Fischer and Gerhard J. Woeginger have been supported by the DFG RTG 2236 "UnRAVeL".

Acknowledgements We thank Bettina Klinz for several helpful discussions about the topic.

1 Introduction

The Assignment Problem (AP) is a fundamental and well-investigated problem in discrete optimization: For the complete bipartite graph $K_{n,n} = (V, E_{n,n})$ with given costs $c : E_{n,n} \to \mathbb{R}$ on the edges, the AP asks for a perfect matching M in $K_{n,n}$ that minimizes the total cost $c(M)$. The AP can be solved in polynomial time, by using for instance the Hungarian method or techniques from network flow theory; see Burkard, Dell'Amico & Martello [3].

[1] RNC[2] is the randomized version of NC[2]. For a definition of NC[2] see [1].

In this paper we study a variant of the AP from the area of robust optimization, which we denote as Recoverable Assignment Problem (RecovAP). An instance of RecovAP consists of two cost functions $c_1, c_2 : E_{n,n} \to \mathbb{R}$ on the edges of $K_{n,n}$ together with an integer bound k. The goal is to find two perfect matchings M_1 and M_2 that minimize the objective value $c_1(M_1) + c_2(M_2)$, subject to the constraint that M_1 and M_2 have at least k edges in common. We also consider the following two non-trivial special cases of RecovAP:

- Consider an arbitrary (bipartite) subgraph $G = (V, E)$ of $K_{n,n}$. If the cost functions c_1 and c_2 are set to $+\infty$ on all edges outside E, one arrives at the graphic special case of RecovAP for bipartite input graphs G. This allows us to study the problem with graph-theoretic tools, and to look into graph-theoretic structures.
- If the cost function c_1 is set to zero on the edges of some fixed perfect matching and set to $+\infty$ on all the remaining edges, the perfect matching M_1 is thereby fixed and frozen at the zero-cost edges. Then problem RecovAP boils down to finding a matching M_2 that minimizes $c_2(M_2)$ subject to the constraint $|M_1 \cap M_2| \geq k$; we denote the resulting optimization problem as Second-stage recoverable assignment problem (2S-RecovAP).

Both problems RecovAP and 2S-RecovAP are motivated by (central and natural) questions in the area of Recoverable Robust Optimization.

Known and related results. The study of discrete optimization problems with intersection constraints (as imposed in problem RecovAP) was initiated through applications in Recoverable Robust Optimization under interval uncertainty. The literature mainly analyzes situations where the feasible solutions form the bases of various types of matroids: Kasperski & Zieliński [14] construct a polynomial time solution for the case of uniform matroids; the underlying robust optimization problem is called the recoverable selection problem. Lachmann, Lendl & Woeginger [15] provide a simple greedy-type algorithm for recoverable selection, and thereby improve the time complexity in [14] from cubic time down to linear time. Hradovic, Kasperski & Zieliński [12, 11] obtain a polynomial time algorithm for the recoverable matroid basis problem and a strongly polynomial time algorithm for the recoverable spanning tree problem. These results have been generalized and improved by Lendl, Peis & Timmermans [16] who show that the recoverable matroid basis and the recoverable polymatroid basis problem can both be solved in strongly polynomial time. Iwamasa & Takayawa [13] further generalize these results and cover cases with nonlinear and convex cost functions.

Büsing [5] derives various NP-hardness results for recoverable robust shortest s-t-path problems, and thus makes one of the first steps in this area beyond feasible solutions with a matroidal structure. Further results about s-t-paths with intersection constraints are obtained by Fluschnik et al. [8]. Note that the combinatorics of s-t-paths is substantially more complex than the combinatorics of matroid bases: whereas all bases of a matroid have the same cardinality, different s-t-paths may contain totally different numbers of edges. For that reason, recoverable robust shortest s-t-path problems do not (easily) translate into corresponding optimization problems that ask for two feasible solutions with at least k common elements.

Şeref et al. [19] study 2S-RecovAP and obtain a randomized algorithm running in polynomial time if the costs are polynomially bounded. A stable matching variant of 2S-RecovAP has recently been introduced and studied by Bredereck et al. [2].

Our contribution. By analyzing problem RecovAP, we take another step beyond matroidal structures in recoverable robust optimization. Section 2 discusses the computational complexity of RecovAP. We look into the parameterized complexity of RecovAP. We show that

the problem is W[1]-hard with respect to the central parameter k, the lower bound on the intersection size of the two matchings. Furthermore, the problem is W[1]-hard with respect to the so-called recoverability parameter $k' = n - k$, hence the problem that asks to have all of the n matching edges of M_1 and M_2 to coincide except for up to k exceptions. These hardness results even hold in the highly restricted case where both cost functions c_1 and c_2 only take the values 0 and 1. Similar W[1]-hardness results hold for the graphic version of RecovAP on planar graphs. On the positive side, there exists a simple XP algorithm for parameter k (that checks all possible sets $M_1 \cap M_2$ of size k) and there also exists a simple XP algorithm for parameter k' (that checks all possible sets $M_1 - M_2$ and $M_2 - M_1$ of size k'). This is in contrast to the variants of the problem with the constraint $|M_1 \cap M_2| \leq k$ or $|M_1 \cap M_2| = k$. These problems are easily shown to be NP-hard for each fixed k via a reduction from the Disjoint Matchings Problem [9]. Finally, we show that the graphic version of RecovAP with respect to parameter treewidth is in FPT.

Next, in Section 3 we discuss problem RecovAP under Monge-type conditions; we refer to Burkard, Klinz & Rudolf [4] for an extensive overview of Monge properties. The cost function in the assignment problem may naturally be viewed as an $n \times n$ cost matrix. If both cost functions c_1 and c_2 correspond to Monge matrices, problem RecovAP boils down to something trivial: In the optimal solution both matchings M_1 and M_2 run along the main diagonal of the underlying matrix. And if both cost functions c_1 and c_2 correspond to Anti-Monge matrices, then in the optimal solution both matchings M_1 and M_2 run along the secondary diagonal of the underlying matrix. The mixed case where c_1 corresponds to a Monge matrix and where c_2 corresponds to an Anti-Monge matrix is less trivial and more interesting. By analyzing the combinatorial structure of potential optimal solutions, we show that it is solvable in polynomial time.

Finally, in Section 4 we turn to the second-stage recoverable assignment problem 2S-RecovAP, which shows a strange and rather unpleasant behavior. We feel that problem 2S-RecovAP is too hard to allow a polynomial time solution, and we simultaneously feel that it is too easy to allow an NP-hardness proof. We support our intuition by two mathematical arguments: First, by a straightforward reduction to the extact matching problem in red blue biparite graphs by Şeref et al. [19], there exists an RNC^2 algorithm for 2S-RecovAP. As the complexity class $\mathrm{RNC}^2 \subseteq \mathrm{RNC}$ is conjectured to be properly contained in NP, this provides evidence for the easiness of 2S-RecovAP. Secondly, we show that the exact matching problem in red-blue bipartite graphs [7] is logspace reducible to 2S-RecovAP. As the existence of a polynomial time algorithm for this exact red-blue matching problem is doubtful (and constitutes a long-open famous problem), this provides evidence for the hardness of 2S-RecovAP.

Due to the page limit, some of the proofs are omitted or a short sketch is given. The detailed proofs will be published in the full version of the paper.

2 Parameterized Complexity

To show W[1]-hardness of the RecovAP problem we reduce from the well known grid tiling problem. In the grid tiling problem we are given an $\ell \times \ell$ grid in which every cell contains a set of tuples. The task is to select a value for every row and for every column compatible with the tuples in the cells: That is, each cell defined by a row and column combination contains a tuple with the values selected for this row and column.

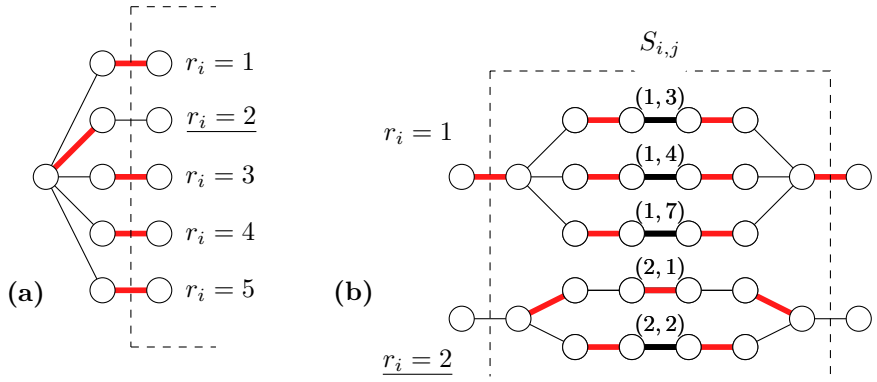

Figure 1 Gadgets for the W[1]-hardness result for RecovAP and parameter k: (a) Selection gadget for row values (analogously for column values). The depicted matching fixes value 2. (b) Component for one row in $S_{i,j}$ (analogously for one column). Each middle edge represents one tuple in $S_{i,j}$ This component exists for both columns and rows and the middle edges are identified if the corresponding tuples are the same.

Grid Tiling

Input: Integers ℓ, n, and a collection $\mathcal{S} = (S_{i,j})_{(i,j)\in[\ell]\times[\ell]}$ with $S_{i,j} \subseteq [n] \times [n]$.

Question: Are there integers $r_1, \ldots, r_\ell$ and $c_1, \ldots, c_\ell$ such that $(r_i, c_j) \in S_{i,j}$ for every $i, j \in [\ell]$?

Grid Tiling has been shown to be $W[1]$-hard for parameter ℓ and has no $f(\ell)n^{o(\ell)}$-time algorithm [6]. For simplicity, we assume that every value $1, \ldots, n$ appears in at least one tuple of $\mathcal{S}$; which can be achieved by renaming the occurring n many values increasingly. That way we ensure that the size of the numbers are polynomial in the size of the input.

By the same reduction, we obtain a lower bound of the runtime assuming the Exponential Time Hypothesis (ETH); for more details on ETH we refer to [6].

▶ **Theorem 1.** *RecovAP is* W[1]*-hard for parameter k with edge cost $(c_1(e), c_2(e)) \in \{(0,0),(0,1),(1,0),(1,1)\}$ for all edges e, and, unless* ETH *fails, it has no $f(k)n^{o(\sqrt{k})}$-time algorithm.*

Proof. Let $\mathcal{I} = (\mathcal{S}, n, \ell)$ be a Grid Tiling instance. We construct a RecovAP instance that asks for matchings M_1 and M_2 of cost 0 and with at least $k = \ell^2$ edges $e \in M_1 \cap M_2$. Hence, only edges with cost $(0,0)$ may be part of $M_1 \cap M_2$. Our construction uses edges with cost $(0,0)$ only to model tuples of grid cells. By our construction, exactly one $(0,0)$ cost edge (r_i, c_j) per grid cell $S_{i,j}$ may be in $M_1 \cap M_2$, which then fixes the selection of that tuple using a cell gadget. We force these tuples to comply with a global selection of row and column values $r_1, \ldots, r_\ell$ and $c_1, \ldots, c_\ell$, using row and column selection gadgets.

In our reduction, we use two types of gadgets, the *row/column selection gadget* and the *cell gadget*. The property of the row/column selection gadget is to encode the selection of one integer per row/column. The role of the cell gadget is to encode the selection of a tuple $(r_i, c_j) \in S_{i,j}$ which is consistent with the selection of the row selection gadget for row i and the column selection gadget for column j. Each of the gadgets will contain a set of special vertices called terminals. These terminals will later be used to connect the gadgets with each other using additional edges.

In the following, we first formally introduce the subgraphs and costs of these gadgets. Next, we combine these gadgets into an instance of RecovAP.

Since we aim for a solution (M_1, M_2) of cost 0, edges of cost 1 are not allowed in the matchings M_1, M_2. Hence, edges with cost $(0,1)$ can only be included in M_1 and edges with cost $(1,0)$ can only be included in M_2. This fact is heavily used in the following arguments.

For the row selection gadget we construct a graph G^{row} which is a star with center vertex v and leaves $t_1, \ldots, t_n$, the terminals of the gadget. All edges of the row selection gadget have cost $(0,1)$. Note, that exactly one of the terminals can be matched with cost 0 using M_1. This matching corresponds to the selected value in the given row. Also, note that in this case the terminal t^r is the unique vertex matched by M_1 and all other terminals must be matched by M_1 to some vertex outside of the gadget. See Figure 1 (a) for an illustration of the row selection gadget, where the gadget is the part left of the dashed box. All the vertices of the row selection gadget will be matched in M_2 using auxiliary vertices introduced at the end of this construction.

Analogously, we construct the column selection gadget as a graph G^{col} with terminals $t^{\text{csel}(c)}$ for all column choices $c \in [n]$. The only difference is, that all edges are assigned cost $(1,0)$ and the selection is determined by the matching M_2.

For the cell gadget we construct a graph $G^{\text{cell}(S)}$, where S is the set of tuples from $[n] \times [n]$ corresponding to the cell. For each tuple $(r,c) \in S$ there exists a special *tuple edge* $e^{\text{tup}(r,c)}$ of cost $(0,0)$ in $G^{\text{cell}(S)}$. For each possible row choice $r \in [n]$ we introduce two terminal vertices $t^{\text{left}(r)}$ and $t^{\text{right}(r)}$. We construct two parts $G^{\text{rcell}(S)}$ and $G^{\text{ccell}(S)}$ handling the row and column selection in the cell (see Figure 1(b) for an illustration of $G^{\text{rcell}(S)}$). $G^{\text{cell}(S)}$ is then defined as the union of $G^{\text{rcell}(S)}$ and $G^{\text{ccell}(S)}$ identifying the common tuple edges.

We begin by introducing the part of the cell gadget that correspond to the row selection, denoted by $G^{\text{rcell}(S)}$. For this part all vertices must be matched by M_1 either inside the gadget or from outside of the gadget if they are terminals. To match these vertices using M_2 we will later introduce auxiliary vertices. The terminals $t^{\text{left}(r)}$ and $t^{\text{right}(r)}$ are part of $G^{\text{rcell}(S)}$ and connected by distinct internally vertex-disjoint paths of length 5 for each tuple $(r,c) \in S$, where $e^{\text{tup}(r,c)}$ is the center edge of this path. All the edges of these paths, except for the tuple edges, are assigned cost $(0,1)$. Note that there might exist r such there is no tuple $(r,c) \in S$. For such r nothing except for the two terminal vertices is added to $G^{\text{rcell}(S)}$. This construction is depicted in Figure 1 (b) inside of the dashed box. Observe that $t^{\text{left}(r)}$ is matched by M_1 from outside of the gadget if and only if $t^{\text{right}(r)}$ is matched by M_1 from outside of the gadget. The only feasible matching in this case adds the second and third edge along the paths from $t^{\text{left}(r)}$ to $t^{\text{right}(r)}$ to M_1. Hence, for such r none of the tuple edges $e^{\text{tup}(r,c)}$ corresonding to a tuple $(r,c) \in S$ can be matched by M_1. But if $t^{\text{left}(r)}$ is not matched by M_1 from outside of the gadget then also $t^{\text{right}(r)}$ cannot be matched from the outside of the gadget and exactly for one tuple $(r,c) \in S$ the tuple edge $e^{\text{tup}(r,c)}$ and the first and last edge along the path connecting $t^{\text{left}(r)}$ to $t^{\text{right}(r)}$ have to be added to M_1. For all other paths the second and third edge have to be added to M_1. This way of selecting tuple edges is illustrated in Figure 1 (b). All the constructed vertices in this paragraph, except those incident to the tuple edges, have to be matched by M_2 using auxiliary vertices.

Analogously, the parts of the cell gadget that correspond to the column selection are denoted by $G^{\text{ccell}(S)}$. For each possible column choice $c \in [n]$ we introduce two terminal vertices $t^{\text{top}(c)}$ and $t^{\text{bottom}(c)}$, analogous to $t^{\text{left}(r)}$ and $t^{\text{right}(r)}$ for the row choices. They are connected to each other via the tuple edges in the same way as the terminals of $G^{\text{rcell}(S)}$. Again by not matching $t^{\text{top}(c)}$ from the outside by M_2 it is enforced that $t^{\text{bottom}(c)}$ is not matched from the outside by M_2 and exactly one tuple edge $e^{\text{tup}(r,c)}$ must be matched by M_2. All the constructed vertices in this paragraph except those incident to the tuple edges will be matched by M_1 using auxiliary vertices.

The cell gadget $G^{\text{cell}(S)}$ is defined as the union of $G^{\text{rcell}(S)}$ and $G^{\text{ccell}(S)}$. The tuple edges and there incident vertices are introduced only once in $G^{\text{cell}(S)}$ and are identified in this union.

In addition we add $2 \cdot (n + |S|)$ auxiliary vertices to $G^{\text{cell}(S)}$. For each $i \in [n]$ we connect $t^{\text{left}(i)}$ with $t^{\text{top}(i)}$ and $t^{\text{right}(i)}$ with $t^{\text{bottom}(i)}$ via a path of length 2, using $2n$ of the auxiliary vertices (see Figure 2 (a)). Observe that for each tuple $(r, c) \in S$ there are 4 vertices connected to the terminals $t^{\text{left}(r)}$, $t^{\text{top}(c)}$, $t^{\text{right}(r)}$ and $t^{\text{bottom}(c)}$ inside $G^{\text{cell}(S)}$. We add a path of length two between the vertex connected to $t^{\text{left}(r)}$ and the vertex connected to $t^{\text{top}(c)}$; and also add a path of length two between the vertex connected to $t^{\text{right}(r)}$ and the vertex connected to $t^{\text{bottom}(c)}$. The cost of the constructed auxiliary edges connected to vertices in $G^{\text{rcell}(S)}$ are set to $(1, 0)$ and the cost of the constructed auxiliary edges connected to vertices in $G^{\text{ccell}(S)}$ are set to $(0, 1)$.

These constructed paths of length 2 can be used to match each of the connected vertices with M_1 or M_2 respectively which also matches the auxiliary vertices with both of the matchings.

Observe that the tuple edges are the only edges in the cell gadget that can both be matched by M_1 and M_2. Hence, if exactly one terminal $t^{\text{left}(r)}$ is not matched by M_1 and exactly one terminal $t^{\text{top}(c)}$ is not matched by M_2 it holds that the cell gadget can contribute one edge to $M_1 \cap M_2$ if and only if $(r, c) \in S$. Using the auxiliary edges all vertices can be matched by both M_1 and M_2.

Now, we are ready to define the instance of RecovAP on a graph G. For each row $i \in [\ell]$ we add two distinct copies of the row gadget $G^{\text{row}}_{i,1} = G^{\text{row}}$, $G^{\text{row}}_{i,2} = G^{\text{row}}$ with terminals $t^{\text{rsel}(r)}_{i,1}, t^{\text{rsel}(r)}_{i,2}$ to G and for each column $j \in [\ell]$ we add two distinct copies of the column selection gadget $G^{\text{col}}_{j,1} = G^{\text{col}}$, $G^{\text{col}}_{j,2} = G^{\text{col}}$ with terminals $t^{\text{csel}(c)}_{j,1}, t^{\text{csel}(c)}_{j,2}$ to G.

For each cell $(i, j) \in [\ell] \times [\ell]$ we add a distinct copy of the cell gadget $G^{\text{cell}}_{i,j} = G^{\text{cell}(S_{i,j})}$ with terminals $t^{\text{left}(r)}_{i,j}$, $t^{\text{right}(r)}_{i,j}$, $t^{\text{top}(c)}_{i,j}$, $t^{\text{bottom}(c)}_{i,j}$ to G.

We now connect the terminals of these gadgets. For each row $i \in [\ell]$ and possible choice r we connect $t^{\text{rsel}(r)}_{i,1}$ to $t^{\text{left}(r)}_{i,1}$ and $t^{\text{rsel}(r)}_{i,2}$ to $t^{\text{right}(r)}_{i,\ell}$ with cost $(0, 1)$. For each column $j \in [\ell]$ and possible choice c we connect $t^{\text{csel}(c)}_{j,1}$ to $t^{\text{top}(c)}_{1,j}$ and $t^{\text{csel}(c)}_{j,2}$ to $t^{\text{bottom}(c)}_{\ell,j}$ with cost $(1, 0)$. For each $i \in [\ell], j \in [\ell - 1]$ and choice r we connect terminals $t^{\text{right}(r)}_{i,j}$ with $t^{\text{left}(r)}_{i,j+1}$ with cost $(0, 1)$. For each $i \in [\ell - 1], j \in [\ell]$ and column choice c we connect terminals $t^{\text{bottom}(c)}_{i,j}$ with $t^{\text{top}(c)}_{i+1,j}$ with cost $(1, 0)$.

To ensure the existence of perfect matchings we add for the vertices of the selection gadgets $2 \cdot (\ell + \ell n)$ additional auxiliary vertices to G. The first half is used to connect the vertices of the row selection gadgets $G^{\text{row}}_{i,1}$ on the left of the grid with the vertices of the column selection gadgets $G^{\text{col}}_{i,1}$ on the top via paths of length 2 for all $i \in [n]$. The others are used to connect the vertices of the row selection gadgets $G^{\text{row}}_{i,2}$ on the right to the column selection gadgets $G^{\text{col}}_{i,2}$ on the bottom via paths of length 2 for all $i \in [n]$. The cost of the edges added in these paths is set to $(1, 0)$ for all edges incident to a vertex of a row selection gadget and $(0, 1)$ for all edges incident to a vertex of a column selection gadget. Hence, all vertices in the row and column selection gadgets can be matched by both matchings with cost 0.

It is now easy to check that the constructed graph is bipartite. To create an instance of RecovAP we add additional edges with cost $(1, 1)$ to obtain a complete bipartite graph. Note that these edges can neither be used by M_1 nor by M_2.

It remains to show that the given instance of the Grid Tiling is a yes-instance if and only if there exists a solution to the constructed instance G of RecovAP with $|M_1 \cap M_2| \geq \ell^2$ of cost 0.

By construction, all the matched tuple edges in each row must be consistent with the row selection and all the matched tuple edges in each column must be consistent with the column selection, else the matchings M_1, M_2 cannot have cost 0. Hence, $|M_1 \cap M_2| \geq \ell^2$ can only be obtained if the same tuple edge is used inside each cell gadget by both the row and column selection.

For the converse direction observe that given a yes-instance of Grid Tiling and the corresponding solution one can easily set the matchings M_1 and M_2 in the constructed gadgets according to the solution for Grid Tiling and obtain a solution for RecovAP of cost 0 such that $|M_1 \cap M_2| \geq \ell^2$.

To show the ETH lower bound assume, for the sake of contradiction, that there is an algorithm for RecovAP with running time $f(k)n^{o(\sqrt{k})}$. Then an instance of Grid Tiling can be transformed in polynomial time into an instance of RecovAP. For the parameter it holds that $k = \ell^2$. So this leads to a running time $f(\ell)n^{o(\sqrt{\ell^2})} = f(\ell)n^{o(\ell)}$. This is a contradiction. ◀

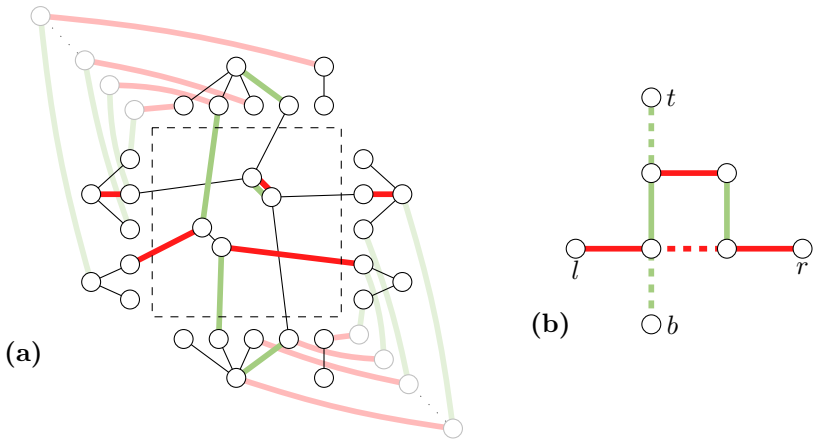

Figure 2 (a) A sketch of the surrounding of a grid cell component (in the dashed box) with matching M_1 in green and M_2 in red. Most of the edges leading into the box are left out, except for two example diagonal $(0,0)$ cost edges, one in both matchings, one in none. Outside in faint color are the additional helper vertices to make the matchings perfect. By positioning these helper vertices on a diagonal as depicted, there are only crossings of edges of cost $(0,1)$ and $(1,0)$. (b) A crossing gadget that replaces a crossing of a $(0,1)$ cost edge $\{l,r\}$ and $(1,0)$ cost edge $\{t,b\}$. The red (dashed and fully drawn) edges have cost $(0,1)$ and the green cost $(1,0)$. The fully drawn red edges show a matching replacing $\{\ell,r\} \in M_1$ while the fully drawn green edges show a matching replacing $\{t,b\} \notin M_2$.

This hardness results also translates to planar graphs with the help of a crossing gadget. Since the input graph is not complete, we no longer need edge cost $(1,1)$.

▶ **Corollary 2.** *RecovAP is* W[1]*-hard on planar graphs for parameter k with $(c_1(e), c_2(e)) \in \{(0,0),(0,1),(1,0)\}$ for all edges e, and unless* ETH *fails it has no $f(k)n^{o(\sqrt{k})}$-time algorithm.*

Proof. The key observation is that by arranging the vertices in the plane as shown in Figure 2(a), there are only crossing edges of cost $(0,1)$ and $(1,0)$. Crossings appear from edges of cost $(0,1)$ in G^{row} and $G^{\text{rcell}(S)}$ with edges of cost $(1,0)$ in G^{col} and G^{ccell}. Also when connecting the auxiliary vertices such crossings appear. Note that when connecting the auxiliary vertices it does not matter which specific vertices are connected. Only the fact that each vertex is connected matters. Hence it is possible to arrange for each cell gadget half of

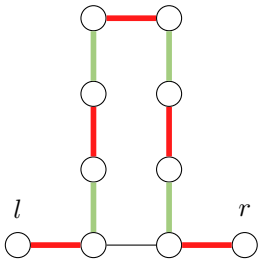

Figure 3 The gadget $G_{l,r}^{\text{snake}(>k,0)}$ that replaces an edge $\{l, r\}$ and effectively simulates an edge of cost $(k + 1, 0)$, for $k = 3$.

the auxiliary vertices in the northwest and half of them in the southeast (see Figure 2(a)). This way it is possible to order the vertices on the left from top to bottom and connect them in this order without any crossings of these $(1, 0)$ cost edges. They then only cross other cost $(0, 1)$ edges of the cell gadget. The principle holds for all other edges connecting the auxiliary vertices of the cell gadget. For the edges connecting auxiliary vertices introduced for the row and column selection gadgets the same idea works by putting them in the northwest and southeast of the whole graph G.

We now introduce the crossing gadget G^{cross} which is used to replace every crossing of an edge $\{\ell, r\}$ of cost $(0, 1)$ with an edge $\{t, b\}$ of cost $(1, 0)$ in G. Graph G^{cross} is illustrated in Figure 2(b). Graph G^{cross} consists of 4 vertices v_1, v_2, v_3, v_4 and the edges $\{\ell, v_1\}$, $\{v_2, r\}$ and $\{v_3, v_4\}$ of cost $(0, 1)$ and the edges $\{t, v_3\}$, $\{v_3, v_1\}$ and $\{v_4, v_2\}$ of cost $(1, 0)$.

Observe that the case $\{\ell, r\} \in M_1$ is simulated by $\{\ell, v_1\} \in M_1$, $\{v_2, r\} \notin M_1$ and $\{v_3, v_4\} \in M_1$ and the case $\{\ell, r\} \notin M_1$ is simulated by $\{\ell, v_1\} \notin M_1$, $\{v_2, r\} \in M_1$ and $\{v_3, v_4\} \notin M_1$. Similarly, there are two ways to simulate $\{t, b\} \in M_2$ and $\{t, b\} \notin M_2$.

It is important that the four new vertices are always matched within this crossing gadget, and thus no further auxiliary vertices are needed. Further, note that this construction can be easily chained in order to handle cases where $\{\ell, r\}$ or $\{t, b\}$ cross more than one other edge. ◀

As a final step regarding planar graphs, we avoid vertices of degree > 4, and we avoid $(0, 0)$ cost edges, thus showing hardness for cost $(0, 1)$ and $(1, 0)$. The key step is to replace a single $(0, 1)$ edge by a long path-like gadget that effectively simulates a cost $(0, k + 1)$ edge, analogously for a $(1, 0)$ edge. Figure 3 shows such a gadget.

▶ **Corollary 3.** *RecovAP is* W[1]*-hard on planar graphs with maximum degree 4 for parameter k with $(c_1(e), c_2(e)) \in \{(0, 1), (1, 0)\}$ for all edges e, and unless* ETH *fails it has no $f(k)n^{o(\sqrt{k})}$-time algorithm.*

Using similar ideas we can also show W[1]-hardness for the dual parameter $k' = n - k$, hence the problem that asks to have all of the n matching edges of M_1 and M_2 to coincide except for up to k exceptions. In robust optimization this parameter is of importance and called the recoverability parameter.

▶ **Theorem 4.** *RecovAP is* W[1]*-hard for the recoverability parameter $k' = n - k$ with $(c_1(e), c_2(e)) \in \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ for all edges e, and, unless* ETH *fails, it has no $f(k)n^{o(\sqrt{k})}$-time algorithm.*

Ideally, Theorem 4 would translate to planar graphs by using a crossing gadget. However, according to Gurjar et al. [10], such a crossing gadget does not exist in this case.

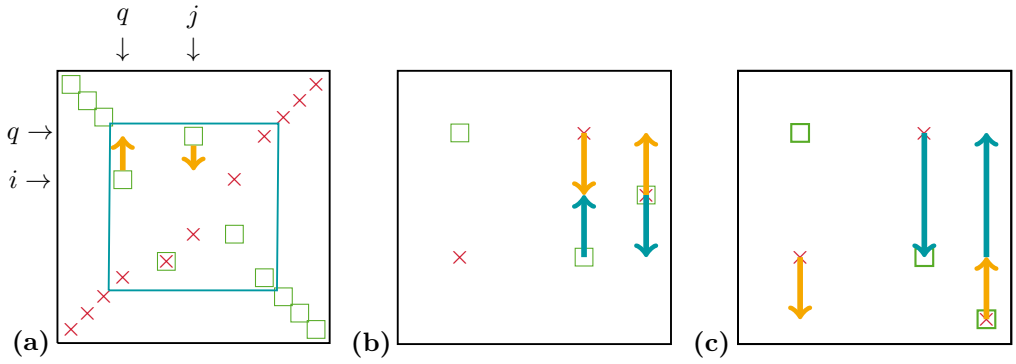

Figure 4 Illustrations for the Monge and Anti-Monge case. (a) Modification to ensure that no cycles have length larger than four. (b) Modification to move 2-cycles east of a 4-cycle into a 4-cycle. (c) Modification to move 2-cycles southeast of a 4-cycle into a 4-cycle.

Beside planar graphs, we also consider graphs of bounded treewidth. RecovAP is fixed parameter tractable in the treewidth of the input graph (without the intersection size k as parameter). Our algorithm is based on dynamic programming over the tree decomposition.

▶ **Theorem 5.** *RecovAP is in* FPT *with respect to the treewidth of the input graph.*

3 Monge and Anti-Monge Matrices

In this section we develop a polynomial time algorithm for the special case of RecovAP if the cost function c_1 is given by a Monge matrix $A = (a_{i,j}) \in \mathbb{R}^{n\times n}$ and the cost function of c_2 is given by an Anti-Monge matrix $B = (b_{i,j}) \in \mathbb{R}^{n\times n}$. The matrix A is called a Monge matrix if for all $i < k$ and $j < l$ it holds that $a_{i,j} + a_{k,l} \leq a_{i,l} + a_{k,j}$. Analogously, the matrix B is called an Anti-Monge matrix if for all $i < k$ and $j < l$ it holds that $b_{i,j} + b_{k,l} \geq b_{i,l} + b_{k,j}$. Let $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$ be the bipartition of the vertex set of $K_{n,n}$. Then the costs of edge $\{u_i, v_j\}$ is given by $c_1(\{u_i, v_j\}) = a_{i,j}$ and $c_2(\{u_i, v_j\}) = b_{i,j}$.

Note, that it is a well-known result that if the cost function of the AP is given by a Monge matrix then the diagonal $\{\{u_i, v_i\} \colon i = 1, \ldots, n\}$ is an optimal solution, similarly the anti-diagonal for Anti-Monge matrices.

Surprisingly, for these special cost functions also the RecovAP has an optimal solution of a similar combinatorial structure, as we show in the following. We illustrate solutions in matrix form by highlighting entry (i, j) with a square if $\{u_i, v_j\} \in M_1$ and with a cross if $\{u_i, v_j\} \in M_2$.

▶ **Theorem 6.** *Let c_1 be given by a Monge matrix A and c_2 be given by an Anti-Monge matrix B. Then, there exists an optimal solution M_1, M_2 to the RecovAP such that*

1. *$\{u_i, v_i\} \in M_1$ for all $i = 1, \ldots, \lfloor \frac{n-k}{2} \rfloor$ and $i = \lceil \frac{n+k}{2} \rceil, \ldots, n$,*
2. *$\{u_i, v_{n+1-i}\} \in M_2$ for all $i = 1, \ldots, \lfloor \frac{n-k}{2} \rfloor$ and $i = \lceil \frac{n+k}{2} \rceil, \ldots, n$,*
3. *$M_1 \cap M_2$ is an optimal solution to the AP with cost $c_1 + c_2$ on the complete bipartite subgraph induced by the sets of vertices $\{u_i \mid i = \lfloor \frac{n-k}{2} \rfloor + 1, \ldots, \lceil \frac{n+k}{2} \rceil - 1\}$ and $\{v_i \mid i = \lfloor \frac{n-k}{2} \rfloor + 1, \ldots, \lceil \frac{n+k}{2} \rceil - 1\}$.*

Based on this structural result we can easily compute an optimal solution for RecovAP by solving the instance of the AP on the subgraph stated in point 3 of Theorem 6 and then completing the perfect matchings M_1 and M_2 as stated in points 1 and 2. In summary, we obtain the following result.

▶ **Theorem 7.** *Let c_1 be given by a Monge matrix A and c_2 be given by an Anti-Monge matrix B. Then the* RecovAP *can be solved in $O(n + k \log k)$ time.*

In the following we prepare the proof of Theorem 6 based on several structural lemmas. The main tool to analyze feasible solutions M_1, M_2 is their decomposition into M_1-M_2-alternating cycles of even length. For any even number s we call such an alternating cycle an s-cycle. Using this language, we call an edge $e \in M_1 \cap M_2$ a 2-cycle. A 4-cycle consists of four edges $\{u_i, v_j\}, \{u_{i'}, v_{j'}\} \in M_1$ and $\{u_i, v_{j'}\}, \{u_{j'}, v_i\} \in M_2$. Note that by the fact that A is Monge and B is Anti-Monge the cost of such edges is minimum if $i < i'$ and $j < j'$. We call a 4-cycle fulfilling this property an aligned 4-cycle. We identify the 4-cycle with its indices (i, j, i', j'). Similarly, we identify a 2-cycle $\{v_i, v_j\} \in M_1 \cap M_2$ with its indices (i, j). Also, observe that in our matrix visualization 4-cycles correspond to 2×2 submatrices where the corners diagonal to each other are marked by squares and crosses. The fact that $i < i'$ and $j < j'$ implies that the squares are drawn into the northwest and southeast corner and the crosses are drawn into the northeast and southwest corners. We say that a 4-cycle (i_1, j_1, i'_1, j'_1) is nested inside another 4-cycle (i_2, j_2, i'_2, j'_2) if it holds that $i_2 < i_1 < i'_1 < i'_2$ and $j_2 < j_1 < j'_1 < j'_2$. Analogously we say that a 2-cycle (i_1, j_1) is nested inside a 4-cycle (i_2, j_2, i'_2, j'_2) if $i_2 < i_1 < i'_2$ and $j_2 < j_1 < j'_2$.

Note, that in the language of such cycles Theorem 6 is equivalent to: there exists an optimal solution which consists of $\lfloor \frac{n-k}{2} \rfloor$ many aligned 4-cycles and all the other matching edges are 2-cycles; all 4-cycles are nested into each other and the 2-cycles are nested inside the innermost 4-cycle; the 2-cycles form a minimum cost perfect matching with respect to $c_1 + c_2$ on their vertices.

In Lemma 8 we prove that there always exists an optimal solution without s-cycles for $s > 4$, and all the 4-cycles are nested and aligned. The main idea here is to iteratively remove such long cycles from the outside to the inside. In a second step (Lemma 9) we then show that there exists an optimal solution in which all 2-cycles lie inside the innermost 4-cycle.

▶ **Lemma 8.** *Let perfect matchings M_1, M_2 be feasible solutions to 2S-*RecovAP*. Then there exists a solution M'_1, M'_2 consisting only of 2-cycles and aligned 4-cycles, and all the 4-cycles are nested.*

Proof. As a first step consider the subinstance (submatrices) where all vertices (rows and columns) contained in 2-cycles are removed (which makes handling row an column indices easier in the following). The resulting submatrices of A and B remain Monge and Anti-Monge.

Now assume that for $q = 1, \ldots, \ell - 1$ it we have that $(q, q, n+1-q, n+1-q)$ already forms a 4-cycle in M_1, M_2. Note that as base of the induction the case $\ell = 1$ trivially true. We now construct matchings M'_1, M'_2 of smaller or equal cost such that $(q, q, n + 1 - q, n + 1 - q)$ is also a 4-cycle for $q = \ell$. If $\{u_q, v_q\} \notin M_1$ it there are edges $\{u_i, v_q\}, \{u_q, v_j\} \in M_1$. We have that $q < i < n + 1 - q$ and $q < j < n + 1 - q$ due to our assumption on present 4-cycles. Because A is Monge, we can exchange those edges for the edges $\{u_q, v_q\}, \{u_i, v_j\}$ in M'_1. See Figure 4 (a) for an illustration of this modification. Analogously, we can ensure that M'_1 also contains $\{u_{n+1-q}, v_{n+1-l}\}$ and M'_2 contains both $\{u_{n+1-q}, v_q\}, \{u_q, v_{n+1-q}\}$, forming the 4-cycle as claimed.

By induction we obtain a solution consisting of only nested aligned 4-cycles, except for maybe one additional 2-cycle exactly in the center of the matrix. We obtain the solution M'_1, M'_2 as claimed by adding back the vertices (rows and columns) of the 2-cycles removed in the first step. ◀

▶ **Lemma 9.** *There is a solution M_1, M_2 with minimum cost $c_1(M_1) + c_2(M_2)$ where all 4-cycles are nested and aligned, and where no 2-cycle is outside of a 4-cycle.*

Proof. Note that the first part of the claim is already implied by Lemma 8, and we start with matchings M_1, M_2 fulfilling the structure stated in Lemma 8. For the second claim we again process the cycles from outside to inside with respect to the nesting order of the 4-cycles. Assume that (x, y) is the outmost 2-cycle in M_1, M_2, and let (i, j, i', j') be the outmost 4-cycle that does not contain (x, y). We show how to modify M_1, M_2 such that the cost does not increase, the number of 2-cycles does not decrease and such that the number of 4-cycles that contain all 2-cycles is increased by one.

We do this by looking at two distinct cases (up to symmetry). Case 1, we have $i < j < x$ and $i' < y < j'$, i.e., (x, y) lies east of (i, j, i', j'). In this case we remove $\{u_i, v_{j'}\}$ and $\{u_x, u_y\}$ from M_2 and add $\{u_x, v_{j'}\}$ and $\{u_i, v_y\}$ to M_2, which can only improve the cost, since B is Anti-Monge. In addition we remove $\{u_{i'}, v_{j'}\}$ and $\{u_x, v_y\}$ from M_1 and add $\{u_x, v_{j'}\}$ and $\{u_{i'}, v_y\}$ to M_1. See Figure 4 (b) for this modification. Note, that now (i, j, i', y) is a new 4-cycle containing the new 2-cycle (x, j'). A 2-cycle lying to the north, south or west of the 4-cycle can be handled symmetrically.

Case 2 is the case when the 2-cycle lies to the southeast of the 4-cycle, i.e. $i < i' < x$ and $j < j' < y$. In this case we remove $\{u_{i'}, v_j\}$ and $\{u_x, v_y\}$ from M_2 and replace it with $\{u_x, v_j\}$ and $\{u_{i'}, v_y\}$ which can only decrease the cost by the fact that B is Anti-Monge. As a second step we remove $\{u_i, v_{j'}\}$ and $\{u_{i'}, v_y\}$ from M_2 and add $\{u_{i'}, v_{i'}\}$ and $\{u_i, v_y\}$ to M_2. See Figure 4 (c) for this modification. Note, that now (i, j, x, y) is a new 4-cycle containing the new 2-cycle (i', j'). The cases when the 2-cycle lies northeast, southwest or northwest can be handled similarly. ◀

Now we are ready to give the proof of Theorem 6.

Proof of Theorem 6. Basically Lemma 9 already implies the combinatorial structure claimed in Theorem 6. The only point missing is that there are exactly $\lfloor \frac{n-k}{2} \rfloor$ many nested 4-cycles in the solution and the remaining edges form 2-cycles inside.

Note that selecting more 2-cycles than strictly necessary (by the constraint or the combinatorial structure) is never helpful, since because of the Monge structure 4-cycles correspond to the optimal solution of the two independent APs.

Hence, if 2 divides $n - k$ the theorem follows directly, there is an optimal solution with k 2-cycles at positions i, j with $\frac{n-k}{2} < i, j < \frac{n-k}{2}$.

Otherwise, if $n - k$ is not multiple of 2, we can assume that n is even and k is odd, since if n is odd an optimal solution always selects the edge $\{u_{(n+1)/2}, v_{(n+1)/2}\}$ as a 2-cycle, giving an equivalent instance with $n - 1$ rows and columns and the constraint to select at least $k - 1$ many 2-cycles. Hence let n be even and k odd. Since the number of edges in a 2-cycle is even, we must select at least $k + 1$ many 2-cycles, and the claim follows. ◀

4 The Second Stage Recoverable Assignment Problem

In this section we study a variant of RecovAP in which the perfect matching M_1 is fixed and we are looking for a perfect matching M_2 of minimum linear cost $c_2(M_2)$ subject to the constraint that $|M_1 \cap M_2| \geq k$. Note that 2S-RecovAP is a special case of RecovAP, with the special cost structure $c_1(e) = 0$ if $e \in M_1$ and $c_1(e) = \infty$ otherwise.

Using the language of recoverable robust optimization this problem is called the incremental assignment problem. Şeref et al. [19] study this problem and obtain a straightforward reduction to Exact Matching in Red-Blue Bipartite Graphs, one of the few natural problems known to be in Randomized-NC (RNC) for which no polynomial time algorithm is known.

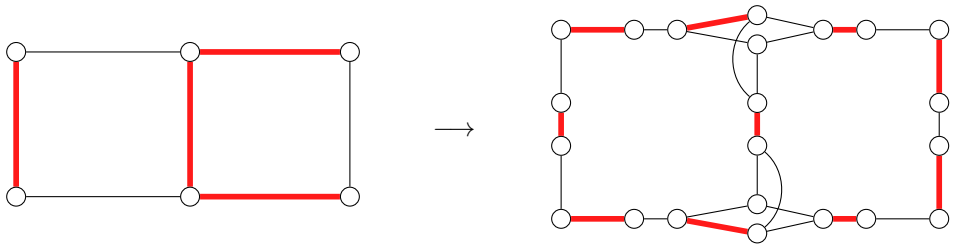

Figure 5 Visualization of the reduction from the special case of Exact Matching in Red-Blue Bipartite Graphs where R is a matching to general Exact Matching in Red-Blue Bipartite Graphs.

Exact Matching in Red-Blue Bipartite Graphs

Input: A bipartite graph $G = (U \cup V, E)$, a subset of edges $R \subseteq E$ (the red colored edges), and an integer $k \in \mathbb{N}$.

Question: Is there a perfect matching M of G such that $|M \cap R| = k$.

Mulmuley et al. [17] show that this problem can be solved in randomized polynomial time if all costs are polynomially bounded. In summary, the following result holds.[2]

▶ **Corollary 10** (Şeref et al. [19]). *2S-RecovAP can be solved by an* RNC^2 *algorithm, if all costs c_2 are polynomially bounded.*

The techniques for this algorithm are not specific to bipartite graphs. Hence they can also be used to solve the second stage of the recoverable perfect matching problem on general graphs in RNC^2.

Surprisingly, we are able to prove that the complexity of these problems is essentially equal. We show that 2S-RecovAP is at least as hard as Exact Matching in red-blue bipartite graphs. Note here that our following logspace reduction implies a reduction in NC^2 [18].

▶ **Theorem 11.** *Exact Matching in Red-Blue Bipartite Graphs is logspace reducible to 2S-RecovAP.*

Proof. As a first part of the proof we give a reduction from exact matching in red-blue colored bipartite graphs to the special case of the problem where the set of red edges forms a matching. In the second part we then show that we can reduce this problem to 2S-RecovAP.

For this first part let $((G = (U \cup V), E), R, k)$ be the given instance of exact matching. We construct a new bipartite graph $G'' = (U'' \cup V'', E'')$ and a set of edges $R'' \subseteq E''$ (see Figure 5 for an illustration). Note that without loss of generality we can assume that G contains no vertex of degree one, since such vertices can always be preprocessed in a trivial way.

For every vertex v in G we add an independent set $v_1, \ldots, v_{\deg(v)-1}$ of $\deg(v) - 1$ many vertices to G''. If $v \in U$ the vertices are added to U'', otherwise if $v \in V$ the vertices are added to V''. For every edge $e = \{u, v\}$ of G we add two vertices u_e and v_e to G'' and connect them by the edge $\{u_e, v_e\}$. If $\{u, v\} \in R$. Then we add $\{u_e, v_e\}$ to R''. In addition we add all the edges $\{u_i, u_e\}$ for $i = 1, \ldots, \deg(u) - 1$ and $\{v_i, v_e\}$ for $i = 1, \ldots, \deg(v) - 1$ to E''. Observe that the graph G'' is bipartite if and only if G is bipartite and note that since $|U| = |V|$ also $|U''| = |V''|$.

[2] For formal definitions of NC^2 and RNC^2 see [18, Sections 15.3 and 15.4].

▷ Claim (a). There exists a perfect matching of G with exactly k edges in R if and only if there exists a perfect matching in G'' with exactly k edges in R''.

Proof. Given a perfect matching M of G with exactly k edges in R we construct a perfect matching M'' in G'' with exactly k edges in R''. For each $e \in M$ we add the edge $\{v_e, u_e\}$ to M''. Note that this way we add exactly k edges from R'' to M''. Now for every vertex v exactly one of its incident edges in G is in M. Hence there are exactly $\deg(v) - 1$ incident edges that are not in M. For each such edge $\{v, w\} \in E \setminus M$ we select one of the vertices v_j for $j \in \{1, \ldots, \deg(v) - 1\}$ and add $\{v_j, v_e\}$ to M''. Note that this way M'' is a perfect matching in G'' with exactly k edges from R'' in M''.

For the converse direction, assume that M'' is a perfect matching of G'' with exactly k edges from R''. We construct a perfect matching M of G with exactly k edges from R. Note that every original vertex $v \in V$ is replaced by an independent set $v_1, \ldots, v_{\deg(v)-1}$ in G''. Each of the vertices $v_1, \ldots, v_{\deg(v)-1}$ is matched to a vertex $v_{e'}$ for an incident edge $e' \in E$. But since there exist only $\deg(v) - 1$ such vertices and v has exactly $\deg(v)$ incident edges in G there exists a unique edge $e = \{v, w\}$ in G for which v_e is not matched to one of $v_1, \ldots, v_{\deg(v)-1}$. Hence v_e must be matched to its only remaining neighbor w_e by M''. Note, that by similar arguments e is also the unique incident edge $e' = \{v, w\}$ in G for which the vertex $w_{e'}$ is not matched to one of the vertices $w_1, \ldots, w_{\deg(w)-1}$. We add the edge $\{v, w\}$ to M. Since v is an arbitrary vertex in G such an edge is added for every v, hence M is a perfect matching in G. Since M'' contains exactly k edges from R'' and for each edge $\{v_e, w_e\}$ in M'' we add the edge $e = \{v, w\}$ to M, also M contains exactly k edges from R. ◁

As a next step we show how to obtain the instance (G', M_1, c_2, k) of 2S-RecovAP such that there exists a perfect matching of G with exactly k edges in R if and only if the optimal value for $(G' = (U' \cup V', E'), M_2, c_2, k)$ is k. The graph G'' constructed above is a subgraph of G'. In addition to that, for each vertex $v \in V''$ that is not matched by R'' we add an additional vertex v' to G' and the edge $\{v, v'\}$. Since G'' is a bipartite graph with $|U''| = |V''|$ and R'' is a matching we can select for each such v' another unique vertex u' and add the edge $\{v', u'\}$ to G'. We define the set R' as the set of edges consisting of R'' and all edges $\{v, v'\}$ for all $v \in \{u \in V'' \colon u \text{ not matched by } R''\}$. Note that R' is a perfect matching in G'. We set $c_2(e) = 1$ for all $e \in R''$ and $c_2(e) = \infty$ for all edges $\{v, v'\}$ where $v \in \{u \in V'' \colon u \text{ not matched by } R''\}$. For all other edges the cost c_2 is equal to 0.

▷ Claim (b). There exists a perfect matching of G with exactly k edges in R if and only if there exists solution to 2S-RecovAP instance (G', M_1, c_2, k) with cost k.

Proof. Assume that there exists a perfect matching M of G with exactly k edges in R. Then by Claim (a) there also exists a perfect matching M'' of G'' with exactly k edges in G''. Based on M'' we define a perfect matching M_2 in G' in the following way. The matching M'' is added to M_2 and hence all vertices in the subgraph G'' of G' are matched. For all the remaining vertices, by the construction above there exists a unique matching consisting of the edges $\{v', u'\}$ which are added to M_2. Note, that $c_2(M_2) = k$.

For the converse direction, assume that M_2 is a perfect matching in G'' with at least k edges from M_1 and cost k. Since only edges in $R' \cap M_1$ have finite cost and $c_2(M_2) = k$ it holds that exactly k edges from R' are contained in M_2. In addition, since none of the edges $\{v, v'\}$ are contained in M_2 it holds that $M_2 \cap E'$ is a perfect matching in G' with exactly k edges from R'. Hence by Claim (a) there exists a perfect matching in G containing exactly k edges from R. ◁

This completes the reduction as claimed in the theorem. Note that this reduction can be implemented using logarithmic space. We just have to process one vertex after another and need to implement a counter counting up to the degree of a vertex. ◀

The case with costs c_2 that are not polynomially bounded remains open. But note, that an RNC algorithm for this problem would imply an RNC algorithm for the special case of obtaining a minimum cost perfect matching, which is a long standing open problem [7].

References

1 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

2 Robert Bredereck, Jiehua Chen, Dušan Knop, Junjie Luo, and Rolf Niedermeier. Adapting stable matchings to evolving preferences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1830–1837, 2020.

3 Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems: Revised Reprint*. SIAM, 2012.

4 Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of Monge properties in optimization. *Discret. Appl. Math.*, 70(2):95–161, 1996. doi:10.1016/0166-218X(95)00103-X.

5 Christina Büsing. Recoverable robust shortest path problems. *Networks*, 59(1):181–189, 2012.

6 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Parameterized algorithms, 2015. URL: http://ebooks.ciando.com/book/index.cfm/bok_id/1960687.

7 Egres. Exact matching in red-blue bipartite graphs. http://lemon.cs.elte.hu/egres/open/Exact_matching_in_red-blue_bipartite_graphs. Accessed: 2020-07-13.

8 Till Fluschnik, Rolf Niedermeier, Carsten Schubert, and Philipp Zschoche. Multistage st path: Confronting similarity with dissimilarity in temporal graphs. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

9 Alan M Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13(2):161–164, 1983.

10 Rohit Gurjar, Arpita Korwar, Jochen Messner, Simon Straub, and Thomas Thierauf. Planarizing gadgets for perfect matching do not exist. In *International Symposium on Mathematical Foundations of Computer Science*, pages 478–490. Springer, 2012.

11 Mikita Hradovich, Adam Kasperski, and Paweł Zieliński. Recoverable robust spanning tree problem under interval uncertainty representations. *Journal of Combinatorial Optimization*, 34(2):554–573, 2017.

12 Mikita Hradovich, Adam Kasperski, and Paweł Zieliński. The recoverable robust spanning tree problem with interval costs is polynomially solvable. *Optimization Letters*, 11(1):17–30, 2017.

13 Yuni Iwamas and Kenjiro Takayawa. Optimal matroid bases with intersection constraints: Valuated matroids, m-convex functions, and their applications. In *Proceedings of the 16th Annual Conference on Theory and Applications of Models of Computation (TAMC 2020), to appear*, 2020.

14 Adam Kasperski and Paweł Zieliński. Robust recoverable and two-stage selection problems. *Discrete Applied Mathematics*, 233:52–64, 2017.

15 Thomas Lachmann, Stefan Lendl, and Gerhard J. Woeginger. A linear time algorithm for the robust recoverable selection problem. *Discret. Appl. Math.*, 303:94–107, 2021. doi:10.1016/j.dam.2020.08.012.

16 Stefan Lendl, Britta Peis, and Veerle Timmermans. Matroid bases with cardinality constraints on the intersection. *arXiv preprint arXiv:1907.04741*, 2019.

17 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

18 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

19 Onur Şeref, Ravindra K. Ahuja, and James B. Orlin. Incremental network optimization: Theory and algorithms. *Operations Research*, 57(3):586–594, 2009. doi:10.1287/opre.1080.0607.