Preprocessing Under Uncertainty

Stefan Fafianie[1], Stefan Kratsch[2], and Vuong Anh Quyen[3]

1 **University of Bonn, Germany**
`fafianie@cs.uni-bonn.de`
2 **University of Bonn, Germany**
`kratsch@cs.uni-bonn.de`
3 **University of Bonn, Germany**
`vuong@cs.uni-bonn.de`

Abstract

In this work we study preprocessing for tractable problems when part of the input is unknown or uncertain. This comes up naturally if, e.g., the load of some machines or the congestion of some roads is not known far enough in advance, or if we have to regularly solve a problem over instances that are largely similar, e.g., daily airport scheduling with few charter flights. Unlike robust optimization, which also studies settings like this, our goal lies not in computing solutions that are (approximately) good for every instantiation. Rather, we seek to preprocess the known parts of the input, to speed up finding an optimal solution once the missing data is known.

We present efficient algorithms that given an instance with partially uncertain input generate an instance of size polynomial in the amount of uncertain data that is equivalent for every instantiation of the unknown part. Concretely, we obtain such algorithms for MINIMUM SPANNING TREE, MINIMUM WEIGHT MATROID BASIS, and MAXIMUM CARDINALITY BIPARTITE MATCHING, where respectively the weight of edges, weight of elements, and the availability of vertices is unknown for part of the input. Furthermore, we show that there are tractable problems, such as SMALL CONNECTED VERTEX COVER, for which one cannot hope to obtain similar results.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases preprocessing, uncertainty, spanning trees, matroids, matchings

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.33

1 Introduction

In many applications we are faced with inputs that are partially uncertain or incomplete. For example, a crucial part of the input, like availability of particular machines or current congestion of some network or road links, may only be available at short notice and may be subject to frequent change. Similarly, we may have to regularly solve instances of some problem that are very similar except for small modifications, e.g., airport gate scheduling when there are only few irregular flights.

A natural approach to this is to come up with solutions that are robust in the sense that they are close to optimal no matter what instantiation the unknown or uncertain part takes. Intuitively it is clear that one cannot hope to always find a solution that is optimal for all instantiations since then the missing parts would always need to be irrelevant. Similarly, if one has to commit to some solution containing uncertain weights/values, then changing these values can in general rule out any good ratio of robustness.

To avoid this issue, in the present work, when given an instance with missing or uncertain information we do not seek to already commit to a solution but to determine how much of the input we can solve or preprocess without knowing the missing or uncertain parts. In

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 33; pp. 33:1–33:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


particular, this approach permits us to still perform computations once the entire input is known/certain. Thus, there is no general argument that would rule out the possibility of finding optimal solutions since we could always do nothing and just keep the instance as is.

Our question is rather, assuming that we always want to get an optimal solution, how much of the certain part of the input do we need to keep (including any other derived information that we could choose to compute). Clearly, we should expect that increasing the amount of uncertain data should drive up the amount of information that we need to keep. Conversely, in many settings the instantiations of some k bits (e.g. presence of certain k edges in a graph) "only" create 2^k different possible instances. Thus, size exponential in the amount of uncertain data is likely to be easy to achieve, e.g., by hardwiring optimal solutions for all instantiations. In contrast, we are interested in spending only polynomial time (too little to precompute an exponential number of solutions) and preprocessing to a size that is polynomial in the amount of uncertain data.

As an easy positive example, consider a road network modeled by a graph $G = (V, E)$ with weights $w\colon E \to \mathbb{R}_{\geq 0}$ capturing the time to travel along the corresponding road. If all weights are given/certain, then we can easily compute a shortest s,t-path. If, say, weights for edges in some set $F \subseteq E$ are not known (yet) or if they are subject to change (e.g. by road congestion) then we cannot for sure determine a shortest path. Moreover, we cannot in general find a path that will be within any bounded factor of the shortest path: If we have to pick one of two parallel edges, then letting the other one have cost ϵ and ours have cost 1 gives ratio $1/\epsilon$ for arbitrary small ϵ. Preprocessing for this setting, however, is straightforward: The final shortest path will consist in some arbitrary way of edges in F and shortest u,v-subpaths containing no edge of F for $u, v \in \{s, t\} \cup V(F)$. The required u,v-paths can be precomputed by taking shortest paths in $G - F$. All distance information can be stored in a smaller graph on vertex set $\{s, t\} \cup V(F)$ by letting weight of $\{u, v\}$ be equal to the length of a shortest u,v-path in $G - F$. The edges of F are then additional parallel edges and the actual shortest s,t-path can be computed once their weights are known. (One may label edges $\{u, v\}$ by the interior vertices of the shortest u,v-path to quickly extract a shortest s,t-path.) Thus, instead of having to find a shortest path in $G = (V, E)$ once all weights are known it suffices to solve the problem on a graph with at most $2 + 2|F|$ vertices.

Our results. We study similar preprocessing questions for several fundamental problems, namely MINIMUM SPANNING TREE, MINIMUM WEIGHT MATROID BASIS, and MAXIMUM CARDINALITY BIPARTITE MATCHING. In the first two problems, the uncertainty lies in the weight of some of the edges of the input graph respectively elements of the ground set of the matroid. The matroid basis problem of course generalizes the MINIMUM SPANNING TREE question but the latter is probably more accessible and uses essentially the same ideas. For MAXIMUM CARDINALITY BIPARTITE MATCHING we study the setting that in the given bipartite graph $G = (L, R; E)$ there are sets of vertices $L_0 \subseteq L$ and $R_0 \subseteq R$ some of which will not be available (but we do not know yet). To some extent, the latter problem can also be handled by the result for MINIMUM WEIGHT MATROID BASIS, but the output would not be an instance of bipartite matching (see end of Section 4). For all three problems, we give efficient algorithms that derive an appropriate form of equivalent instance such that an optimal solution can be found using just this instance plus the missing input data. Finally, we show that there are problems for which we cannot find efficient compressions that capture all possible scenarios, even if the running time of the algorithm is allowed to be unbounded; these are SMALL CONNECTED VERTEX COVER and some LP-related problems.

Related work. The effect of uncertainty in instances on optimal solutions has long been considered and there are several approaches to deal with it. An early approach is *stochastic optimization* (SO), starting at least from Dantzig's original paper [4], which assumes that the uncertainty has a probabilistic description. A more recent approach to optimization under uncertainty is *robust optimization* (RO). In contrast to SO, the uncertainty model in RO is not stochastic, but rather deterministic and set-based. More precisely, in RO, we want to find a solution to optimize the value of a *certain* function which subjects to a list of *uncertain* constraints. Each of these constraints may depend on some uncertain parameters whose values are in a given domain which may be infinitely large or continuous. The solution is required to satisfy every constraint for *all* values of its uncertain parameters. We refer interested readers to a survey [2] for more information about RO. Both of the two approaches (SO and RO) only work with the uncertainty of the *value* of parameters in an instance, but neither with the uncertainty of *appearance* of any factor in an instance (variables, constraints, vertices, edges, etc.), nor with the uncertainty of the objective function. Moreover, the focus is on finding optimal or approximate solutions rather than preprocessing.

Concerning preprocessing, a related concept is that of *kernelization* from parameterized complexity. In brief, a kernelization for a *decision* problem is an efficient algorithm that compresses each input instance to an instance with smaller size (if possible) and ensures that the answer does not change. Note that the target of this approach is NP-hard problems and sizes of compressed instances are measured by some problem-specific parameter instead of the input size since one cannot hope to efficiently shrink all inputs of some NP-hard problem, unless $\mathsf{P} = \mathsf{NP}$. To the best of our knowledge there has not been much research in this area regarding uncertainty or robustness. Two recent results on kernelization nevertheless use intermediate results that are in line with the present work, and that have in part inspired it:

(1) A nice result of Pilipczuk et al. [13] is the following: Given a plane graph G with outer face B, one can efficiently compress the inner part of G to obtain a smaller graph H such that H contains an optimal Steiner tree connecting terminal set S for *every* subset $S \subseteq B$ of the outer face. Note that for any fixed set $S \subseteq B$ this is a polynomial-time problem, but there are of course $2^{|B|}$ many possible sets S. Pilipczuk et al. use this result to obtain polynomial kernels for several problems on planar graphs, e.g., PLANAR STEINER TREE.

(2) Kratsch and Wahlström [8] obtained the following result on cut-covering sets: Given a graph and two vertex sets S and T, there exists a small vertex set Z such that Z contains a minimum vertex (A, B)-cut for *every* $A \subseteq S$ and $B \subseteq T$, moreover Z can be computed in randomized polynomial time with small error probability. Again, for each choice of A and B this is polynomial-time solvable, but there is an exponential number of choices. We will make use of this result in Section 5 and, furthermore, it directly yields another positive example for the MIN CUT problem with uncertain vertices: Given a graph $G = (V, E)$ with two vertices $s, t \in V$ and $U \subseteq V \setminus \{s, t\}$, we can apply the result for $S = U \cup \{s\}$ and $T = U \cup \{t\}$ to compute a cut-covering set Z. Now, for every $U' \subseteq U$, the set Z contains a minimum $U' \cup \{s\}$, $U' \cup \{t\}$ cut C. Clearly, $U' \subseteq C$ and thus $C \setminus U'$ must be a minimum (s, t)-cut in $G - U'$. This observation together with a technique called *torso operation* (see [10]), which will be explained in Section 5, allows us to compress the certain part of G efficiently. By Menger's theorem, this carries over to the problem of computing the maximum number of vertex-disjoint paths between two vertices in a graph. Thus, one can also obtain a preprocessing for MAX FLOW when all capacities are small, in the sense that the guaranteed size depends polynomially on the maximum capacity.

(3) In [1] Assadi et al. also considered MAXIMUM MATCHING and MINIMUM SPANNING TREE with a similar approach but their model, which they called "The dynamic sketching

model", has two main differences: First, it only captures the uncertainty of *appearance* of edges but not their weights. Second, output of an algorithm in the model may be only a compact structure ("a sketch" in their words) but not an instance of the considered problem.

Other somewhat related paradigms are *online algorithms* and *dynamic algorithms*. In the former, the input is revealed piece-by-piece and the algorithm needs to commit to decisions without knowing the remaining input; the goal is to optimize the ratio between the online solution and an offline optimum. This is quite different from our setting because it requires to commit to a solution. In the dynamic setting a complete instance is given but modifications to it are given in further rounds; the goal is to adapt quickly to the modifications and to find a solution for the modified instance (faster than computing from scratch). This is closer to our setting, by allowing a different solution in each round, but differs by having a complete input in each round and not necessarily restricting the parts of the input that may change.

The problem of finding a minimum spanning tree when edge weights are uncertain has also been explored in a different setting [7, 11]. In this case, all edge weights fall in prespecified intervals and there is a cost for finding out the weight of a specific edge. The goal is to find a cost-efficient query strategy for finding a minimum spanning tree.

Organization. We will start with preliminaries in Section 2 and consider MINIMUM SPANNING TREE in Section 3 as a warm-up. We present our algorithms for MINIMUM WEIGHT MATROID BASIS and MAXIMUM CARDINALITY BIPARTITE MATCHING in Sections 4 and 5 respectively. Finally, we present our lower bounds in Section 6. Proofs omitted in this extended abstract can be found in [6].

2 Preliminaries

Graphs. We mostly follow graph notation as given by Diestel [5]. A *walk* in a graph G is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that for every $i = 0, \ldots, k-1$ the vertices v_i and v_{i+1} are adjacent; if G is a directed graph then it is required that there is an arc directed from v_i to v_{i+1}. A *path* in G is a walk $(v_0, v_1, \ldots, v_k)$ such that v_i and v_j are distinct for every $i \neq j$. In this case, v_0 and v_k are called the first vertex and the last vertex of the path respectively; all other vertices are called *internal vertices*. A (u, v)*-path* is a path whose first vertex is u and whose last vertex is v. If S is a vertex set of a graph G, then we denote by $G - S$ the graph obtained from G by removing all vertices in S and their incident edges.

Matroids. *A matroid* is a pair $(E, \mathcal{I})$, where E is a finite set of elements, called *ground set*, and $\mathcal{I}$ is a family of subsets of E which are called *independent sets* such that: (1) $\emptyset \in \mathcal{I}$. (2) If $A \in \mathcal{I}$, then for every subset $B \subseteq A$ we have $B \in \mathcal{I}$. (3) If A and B are two independent sets in $\mathcal{I}$ and $|A| > |B|$, then there is an element $e \in A \setminus B$ such that $B \cup \{e\} \in \mathcal{I}$; this is called the *augmentation property*.

By the augmentation property, all (inclusion-wise) maximal independent sets have the same cardinality; each of them is called *a basis*. The (inclusion-wise) minimal dependent sets are called *circuits*. Given a matroid $\mathcal{M} = (E, \mathcal{I})$ and $F \subseteq E$, we denote by $\mathcal{M} - F$ the matroid obtained from $\mathcal{M}$ by deleting elements in F, i.e., the matroid on ground set $E \setminus F$ whose independent sets are the independent sets of $\mathcal{M}$ that are disjoint from F. If F is an independent set of $\mathcal{M}$ then we denote by $\mathcal{M}/F$ the matroid obtained from $\mathcal{M}$ by contracting F, i.e., the matroid on ground set $E \setminus F$ such that a set I is independent if and only if $I \cup F$ is an independent set of $\mathcal{M}$. A matroid $\mathcal{M}'$ obtained from $\mathcal{M}$ by a sequence of deletion and contraction operations is called a *minor* of $\mathcal{M}$.

A matrix M over a field gives rise to a matroid $\mathcal{M}$ whose ground set is the set of columns of M and a set of columns is an independent set of $\mathcal{M}$ if and only if it is linearly independent as a set of vectors. In this case, we say that $\mathcal{M}$ is *represented by* M or M is a *representation matrix* of $\mathcal{M}$. Note that there may be different representation matrices of the same matroid and there exist matroids which cannot be represented over any field.

Because there can be as many as $2^{|E|}$ independent sets in a matroid $\mathcal{M} = (E, \mathcal{I})$, to achieve time polynomial in $|E|$ it is necessary to use a more succinct representation, rather than listing all sets in $\mathcal{I}$ explicitly. Two common ways are representing $\mathcal{M}$ by a matrix (not possible for all matroids) or assuming that an independence oracle for $\mathcal{I}$ is provided:

(i) If our matroid is given by a representation matrix over some field, the output of our algorithm should again be a representation matrix. It is known that a representation for any minor of $\mathcal{M}$ can be computed in polynomial time from a representation of $\mathcal{M}$ (cf. Marx [9]).

(ii) If our matroid is given by a ground set and an independence oracle, i.e., a blackbox algorithm which tells us whether an arbitrary subset of the ground set is independent or not, then the time for the oracle is not taken into account. In this case, the output should again be a ground set together with an oracle. Since the oracle is blackbox, the output oracle will be a frontend to the initial oracle and make queries to it.

Further notation. Given two functions $f_1 \colon X_1 \to \mathbb{N}$ and $f_2 \colon X_2 \to \mathbb{N}$ with $X_1 \cap X_2 = \emptyset$, the *union* of f_1 and f_2, denoted by $f_1 \cup f_2$, is the function $f \colon X_1 \cup X_2 \to \mathbb{N}$ defined by $f(x) = f_i(x)$ if $x \in X_i$ for $i = 1, 2$. For convenience, we use $+$ and $-$ instead of $\cup$ and $\setminus$ for singleton sets, e.g., $S + e$ and $S - e$ instead of $S \cup \{e\}$ and $S \setminus \{e\}$. We also abuse notation by using $f(e)$ to represent a function f whose domain is $\{e\}$, e.g., we write $f(e) \cup g$ to clarify that we take the union of two functions f and g where the function f has a singleton domain.

3 Minimum Spanning Tree in Graphs With Some Unknown Weights

For a connected graph $G = (V, E)$ and a weight function $\omega \colon E \to \mathbb{N}$ one can efficiently compute a spanning tree of minimum total edge weight. If, however, the weight of some edges is not known (yet), then in general we cannot (yet) solve the instance. Nevertheless, we may be able to preprocess the known part of the instance in order to save time later.

Say we are given a connected graph $G = (V, E \cup F)$ and a weight function $\omega_E \colon E \to \mathbb{N}$. Over different choices of weights $\omega_F \colon F \to \mathbb{N}$ for edges in F there may be an exponential number of different minimum weight spanning trees. We will show how to efficiently generate a new instance on which we may solve the problem for any instantiation of ω_F, i.e., computing the weight of a minimum spanning tree relative to weights $\omega = \omega_E \cup \omega_F$.

In slight abuse of notation we assume that edges that are originally in F can be identified in the new instance after the operations (edge contractions) performed in our algorithm. That is, given an edge in F, we can find the corresponding edge in the new instance even if the endpoints of this edge have changed. More formally this could also be captured by an appropriate bijection from F to a set of edges F' that appear in the new instance.

▶ **Theorem 1.** *There is a polynomial-time algorithm that, given a connected graph $G = (V, E \cup F)$ and a weight function $\omega_E \colon E \to \mathbb{N}$, computes a connected graph $G' = (V', E' \cup F)$ with $|E'| \leq |F|$, a weight function $\omega_{E'} \colon E' \to \mathbb{N}$, and $k \in \mathbb{N}$, such that, for any $\omega_F \colon F \to \mathbb{N}$, the graph G has minimum spanning tree weight l relative to $\omega_E \cup \omega_F \colon E \cup F \to \mathbb{N}$ if and only if G' has minimum spanning tree weight $l' = l - k$ relative to $\omega_{E'} \cup \omega_F \colon E' \cup F \to \mathbb{N}$.*

Let MSF denote a minimum weight spanning forest of $G - F$ and let $G_1 = (V, \mathrm{MSF} \cup F)$.

The following lemma shows that the weight of a minimum spanning tree in G_1 is equal to that of a minimum spanning tree in G for any weight function on F.

▶ **Lemma 2.** *For any weight function $\omega_F\colon F \to \mathbb{N}$, there is a minimum spanning tree MST_{ω_F} in $(G, \omega_E \cup \omega_F)$ such that $\mathrm{MST}_{\omega_F} \subseteq \mathrm{MSF} \cup F$.*

Accordingly, the first part of the simplification of (G, ω_E) consists of replacing G by G_1 and restricting ω_E to the edges of G_1, i.e., to the edges of $\mathrm{MSF} \cup F$.

Let $\omega_0\colon F \to \mathbb{N}\colon f \mapsto 0$. If an edge $e \in E$ is used by a minimum spanning tree for $(G, \omega_E \cup \omega_0)$ in which all edges in F have zero weight, then, intuitively, using e is also a good choice for spanning trees with other weight functions $\omega_F\colon F \to \mathbb{N}$.

▶ **Lemma 3.** *Let MST_{ω_0} a minimum spanning tree for $(G, \omega_E \cup \omega_0)$. For every $\omega_F\colon F \to \mathbb{N}$, there is a minimum spanning tree MST_{ω_F} for $(G, \omega_E \cup \omega_F)$ that uses all edges in $\mathrm{MST}_{\omega_0} \setminus F$.*

It follows that we can further simplify G_1 by contracting edges of $\mathrm{MST}_{\omega_0} \setminus F$, since for any weight function $\omega_F\colon F \to \mathbb{N}$ there is a minimum spanning tree that uses these edges.

▶ **Lemma 4.** *Let MST_{ω_0} be a minimum spanning tree in $(G, \omega_E \cup \omega_0)$ and let $e \in \mathrm{MST}_{\omega_0} \setminus F$. Let the graph $G_2 = (V', E')$ be obtained by contracting e, where $E' = E - e$, and let $\omega_{E'}$ be ω_E restricted to E'. For any weight function $\omega_F\colon F \to \mathbb{N}$, $(G, \omega_E \cup \omega_F)$ has a spanning tree with weight l if and only if $(G_2, \omega_{E'} \cup \omega_F)$ has a spanning tree MST'_{ω_F} of weight $l - \omega_E(e)$.*

We compute a minimum spanning tree MST_{ω_0} for $(G_1, \omega_E \cup \omega_0)$ and create a graph $G' = (V', E' \cup F')$ by contracting every edge in $\mathrm{MST}_{\omega_0} \setminus F$. Let k be the combined weight of the contracted edges. All edges in E' correspond to edges E after the contractions. We define ω'_E accordingly. Note that $|E'| \leq |F|$ since at most $|\mathrm{MSF}| \leq n - 1$ edges of E remain in G_1, of which we contract at least $|\mathrm{MST}_{\omega_0} \setminus F| \geq |\mathrm{MST}_{\omega_0}| - |F| = n - 1 - |F|$ edges in order to obtain G'. As a result of Lemmas 2 through 4 we have that G', ω'_E, and k satisfy the required properties in Theorem 1 and the result follows.

4 Minimum Weight Basis in Matroids With Some Unknown Weights

Given a matroid $\mathcal{M} = (E, \mathcal{I})$, we know that for each fixed weight function $w\colon E \to \mathbb{N}$ we can find a minimum weight basis of $\mathcal{M}$ in polynomial time by the greedy algorithm. Now suppose that there is a subset $F \subseteq E$ of elements with unknown weights, i.e., we are only given a partial weight function $w\colon E \setminus F \to \mathbb{N}$. We want to reduce the known part of the input such that when given any weights for elements in F we can compute a minimum weight basis.

In the rest of this section, we prove the following result:

▶ **Theorem 5.** *There is a polynomial-time algorithm that given a matroid $\mathcal{M} = (E, \mathcal{I})$, by matrix representation or independence oracle, together with a set $F \subseteq E$ and a partial weight function $w\colon E \setminus F \to \mathbb{N}$, outputs a matroid $\mathcal{M}' = (E', \mathcal{I}')$, a partial weight function $w'\colon E' \setminus F \to \mathbb{N}$, and a number $k \in \mathbb{N}$ such that: (1) E' contains F and has at most $2|F|$ elements. (2) For every weight function on F, the matroid $\mathcal{M}$ has a minimum weight basis of weight l if and only if $\mathcal{M}'$ has a minimum weight basis of weight $l - k$.*

We start by recalling some basics about the interplay of circuits and bases in a matroid.

▶ **Lemma 6** (cf. Oxley [12, Corollary 1.2.6])**.** *Let $\mathcal{M} = (E, \mathcal{I})$ a matroid and $B \in \mathcal{I}$ a basis of $\mathcal{M}$. For every $e \in E \setminus B$ there is a unique circuit C in $B + e$, and this circuit contains e. Moreover, for every $e' \in C - e$, the set $B + e - e'$ is also a basis of $\mathcal{M}$.*

The next lemma is about the relation between minimum weight bases of a given matroid $\mathcal{M}$ and the ones in a sub-matroid $\mathcal{M}'$ of $\mathcal{M}$.

▶ **Lemma 7.** *Let $\mathcal{M} = (E, \mathcal{I})$ a matroid and let $F \subseteq E$. For every weight function, if I is a minimum weight basis of $\mathcal{M} - F$, then there is a minimum weight basis of $\mathcal{M}$ that is contained in $I \cup F$.*

Given a matroid with a non-empty subset F of elements, there are infinitely many weight functions on F. However, since we are considering a minimization problem, there is a special one: the weight function which assigns value zero to every element in F. Intuitively, because elements in F have "cheapest cost" in this case, if an element not in F appears in a minimum weight basis with respect to this weight function then it should also appear in some minimum weight basis with respect to other weight functions. The next lemma verifies this intuition.

▶ **Lemma 8.** *Let $\mathcal{M} = (E, \mathcal{I})$ a matroid, let $F \subseteq E$, and let $\hat{w} \colon E \setminus F \to \mathbb{N}$. If I_0 is a minimum weight basis of $\mathcal{M}$ subject to $w_0 = \hat{w} \cup \hat{w}_0$ where $\hat{w}_0 \colon F \to \mathbb{N} \colon f \mapsto 0$, then for every weight function $w_F \colon F \to \mathbb{N}$ there is a minimum weight basis of $\mathcal{M}$ subject to $w = \hat{w} \cup w_F$ that contains $I_0 \setminus F$.*

Now we describe our algorithm. Given a matroid $\mathcal{M} = (E, \mathcal{I})$ together with a subset $F \subseteq E$ and a partial weight function $w \colon E \setminus F \to \mathbb{N}$, we compute $\mathcal{M}''$ as follows:

1. Compute a minimum weight basis B of $\mathcal{M} - F$.
2. Compute $\mathcal{M}' = \mathcal{M}[B \cup F]$. If $\mathcal{M}$ is given by a representation matrix M, then $\mathcal{M}'$ can be represented by the matrix M' obtained from M by taking only the columns corresponding to the elements in $B \cup F$. If $\mathcal{M}$ is given by an oracle O, then an oracle O' for $\mathcal{M}'$ can be obtained easily: Given a set I, first check whether I is a subset of $B \cup F$, else return no. Query O for whether I is independent in $\mathcal{M}$ and return the answer.
3. Compute a minimum weight basis B_0 of $\mathcal{M}'$ corresponding to the weight function $w_0 \colon E \to \mathbb{N}$ with $w_0(e) = 0$ for all $e \in F$ and $w_0(e) = w(e)$ for $e \in E \setminus F$.
4. Compute $\mathcal{M}'' = \mathcal{M}'/(B_0 \setminus F)$ and $k = w(B_0 \setminus F)$. If $\mathcal{M}'$ is represented by a matrix M', then a matrix representation for $\mathcal{M}''$ can be derived from M' in polynomial time (cf. [9]). If $\mathcal{M}'$ is given by an oracle O', then an oracle O'' for $\mathcal{M}''$ can be obtained as follows: Given a set I first check whether I is a subset of the ground set of $\mathcal{M}''$, else return no. Query O' for whether $I \cup (B_0 \setminus F)$ is independent in $\mathcal{M}'$ and return the answer.

It is easy to see that our algorithm runs in polynomial time. Because B_0 is a basis in $\mathcal{M}'$ and B is an independent set in $\mathcal{M}'$, we have $B_0 \subseteq B \cup F$ and $|B| \leq |B_0|$. The number of elements of $\mathcal{M}'$ not in F is

$$|B \setminus B_0| \leq |(B \cup F) \setminus B_0| = |B \cup F| - |B_0| = |B| + |F| - |B_0| \leq |F|.$$

The final lemma, about the correctness of our algorithm, finishes the proof of Theorem 5.

▶ **Lemma 9.** *For every weight function, the minimum weight of a basis in $\mathcal{M}$ is l if and only if the minimum weight of a basis in $\mathcal{M}''$ is $l - k$.*

Our result can also be applied for the case of maximum weight basis with one additional condition: There is a fixed upper bound for all weights, i.e., we may not know weights of elements in F but we do know that they cannot be larger than some constant c. In that case, if for each element e of $\mathcal{M}$, we replace its weight $w(e)$ by $w'(e) = c - w(e)$ then for every basis I we have $w'(I) = c \cdot |I| - w(I)$. Because all bases in a matroid have the same size, a maximum weight basis with respect to the original weight function must be a minimum weight basis with respect to the new one.

Theorem 5 can be applied for several specific matroid classes. For example, Theorem 1 can be obtained by an application to the class of graphic matroids, noting that these are closed under deletion and contraction. We finish this section discussing an application for *transversal matroids*. Given a bipartite graph $G = (L, R; E)$ the transversal matroid $\mathcal{M} = (R, \mathcal{I})$ has as its independent sets exactly those subsets of R that have a matching into L (equivalently, that are the endpoints of some bipartite matching). If we assign weight 1 for every element of the matroid then the weight of a maximum weight basis in the matroid is the size of a maximum matching in the original bipartite graph. Uncertain vertices in R can be easily simulated by making their weights be uncertain and using weight 0 to mean that they are not available. Observe that we have an upper bound for uncertain weights, so by the above arguments, we can apply the result for maximum weight basis to compress our uncertain instance for maximum matching in bipartite graph. Uncertain vertices in L can be handled by giving each a private neighbor that has uncertain weight: We can set these weights very high (and adjust the target weight of the basis that we are looking for) to enforce that the corresponding uncertain vertex is used to match the private neighbor, thereby preventing a matching with other R vertices. If the vertex is available then set the weight of this private neighbor to 0. However, the output would not be an instance of bipartite matching. Unlike graphic matroids, transversal matroids are not closed under contraction (which would give the larger class of gammoids), and thus it seems unlikely that one could directly extract an appropriate graph. We address this by studying the bipartite matching problem with uncertain vertices directly in the following section, using other techniques.

5 Maximum Matching in Bipartite Graphs With Uncertain Vertices

Given a bipartite graph $G = (L, R; E)$, a maximum matching of G can be found in polynomial time. Now suppose that there are vertex subsets $L_0 \subseteq L$ and $R_0 \subseteq R$ and some arbitrary vertex sets $L' \subseteq L_0$ and $R' \subseteq R_0$ may not be available in the final input, i.e., we will be asked for a maximum matching in $G - (L' \cup R')$. Thus, there are $2^{|L_0|+|R_0|}$ possible instances. How much can we simplify and shrink G when knowing only L_0 and R_0, but not L' and R'? We show that, despite the exponential number of possible final instances, a graph G' with polynomial in $|L_0| + |R_0|$ many vertices is sufficient.

▶ **Theorem 10.** *There is a randomized polynomial-time algorithm that, given a bipartite graph $G = (L, R; E)$, $L_0 \subseteq L$, and $R_0 \subseteq R$, returns a bipartite graph G' with $\mathcal{O}((|L_0|+|R_0|)^4)$ vertices and $k \in \mathbb{N}$ such that for any $L' \subseteq L_0$ and $R' \subseteq R_0$, the graph $G - (L' \cup R')$ has maximum matching size l if and only if $G' - (L' \cup R')$ has maximum matching size $l' = l - k$.*

Let us first recall the concept of *augmenting paths*.

▶ **Definition 11.** Let M a matching in a G. An M-augmenting path is a path in G s.t.
(i) the first and last vertices of the path are not incident to any edge in M and
(ii) edges on the path are alternatingly in M and not in M.

It is well known that if an augmenting path P exists, then we can obtain a matching from M that has one more edge by replacing in M the matched edges on P with the non-matched edges on P. This extends in a natural way to packings of vertex-disjoint augmenting paths.

▶ **Lemma 12.** *Let G be a graph, let M be any matching in G, let M_0 be a maximum matching in G, and let r denote the maximum number of vertex-disjoint M-augmenting paths in G. We have that $r = |M_0| - |M|$.*

We now fix a maximum matching M in $G - (L_0 \cup R_0)$ and use it in the remainder of the section. We direct the edges of G to obtain a directed bipartite graph $H = (L, R; A)$ as follows: Every edge in M is directed from R to L and every edge not in M is directed from L to R. This type of directed graph is part of a folklore approach for finding augmenting paths. Let F_L (resp. F_R) denote the vertices of $L \setminus L_0$ (resp. $R \setminus R_0$) which are not covered by M, and note that $V(M)$, L_0, R_0, F_L, and F_R are pairwise disjoint.

▶ **Observation 1.** For any $L' \subseteq L_0$ and $R' \subseteq R_0$, there is a one-to-one correspondence between directed paths in $H - (L' \cup R')$ from $F_L \cup (L_0 \setminus L')$ to $F_R \cup (R_0 \setminus R')$ and M-augmenting paths in $G - (L' \cup R')$. This is because $F_L \cup (L_0 \setminus L')$ and $F_R \cup (R_0 \setminus R')$ are exactly the vertices that are free in the matching, while a directed path must visit matched edges alternatingly, since these are exactly the edges from R to L.

▶ **Observation 2.** For any $L' \subseteq L_0$ and $R' \subseteq R_0$, there is no M-augmenting path in $G - (L' \cup R')$ that starts in F_L and ends in F_R. This follows from maximality of M, since such a path could be used to obtain a bigger matching in $G - (L_0 \cup R_0)$. By Observation 1 we have that there is no directed path in $H - (L' \cup R')$ from F_L to F_R.

Given the relation between augmenting paths and directed paths we now consider minimum cuts in the graph H. The following theorem of Kratsch and Wahlström [8] provides a small set of vertices that contains minimum cuts for a specified type of requested A, B-cuts.

▶ **Theorem 13** (Kratsch and Wahlström [8])**.** *Let $G = (V, E)$ be a directed graph and let $S, T \subseteq V$. Let r denote the size of a minimum (S, T)-vertex cut (which may intersect S and T). There exists a set $X \subseteq V$ of size $|X| = \mathcal{O}(|S| \cdot |T| \cdot r)$ such that for any $A \subseteq S$ and $B \subseteq T$ the set X contains a minimum (A, B)-vertex cut. Such a set X can be computed in randomized polynomial time with error probability $\mathcal{O}(2^{-n})$.*

The following lemma adapts Theorem 13 to our application, mainly taking care not to inflate the cut size overly much. (We ask for minimum $(F_L \cup (L_0 \setminus L'), F_R \cup (R_0 \setminus R'))$-vertex cuts, but without having size $\Omega(|F_L| + |F_R|)$.)

▶ **Lemma 14.** *There exists a set $X \subseteq L \cup R$ of size $|X| = \mathcal{O}((|L_0| + |R_0|)^3)$ such that for any $L' \subseteq L_0$ and $R' \subseteq R_0$, X contains a minimum $(F_L \cup (L_0 \setminus L'), F_R \cup (R_0 \setminus R'))$-vertex cut in $H - (L' \cup R')$; it can be found in randomized polynomial time with error probability $\mathcal{O}(2^{-n})$.*

▶ **Definition 15.** Let $D = (V, A)$ a directed graph and $Z \subseteq V$. By applying to D the *torso operation* on Z we mean to derive a new graph, denoted by $\text{torso}(D, Z)$, by adding to $D[Z]$ an arc (u, v) for every pair $u, v \in Z$ if there is a directed (u, v)-path in D with no internal vertices from Z. If an arc of $\text{torso}(D, Z)$ is not in $D[Z]$, then we call it a *shortcut arc*.

We now construct a set Z, starting from X as obtained from Lemma 14: Let M_X be the set of edges in M with at least one endpoint in X and let $X' = X \cup L_0 \cup R_0 \cup V(M_X)$; note that $X' \cap (F_L \cup F_R) = \emptyset$. For each $v \in X' \cap R$ (resp. $v \in X' \cap L$), let $F_v \subseteq F_L$ (resp. $F_v \subseteq F_R$) denote the set of vertices that can be reached from v (resp. can reach v) by a directed path in H with no internal vertices from X'. If $|F_v| \leq |L_0| + |R_0|$, then let $W_v = F_v$; otherwise let W_v be an arbitrary subset of F_v of size $|L_0| + |R_0|$. Finally, let $Z = X' \cup \bigcup_{v \in X'} W_v$.

Let $H' = \text{torso}(H, Z)$. By construction there is no matching edge in M with exactly one vertex in Z, which ensures that H' is also a bipartite graph: By construction, a directed path connecting two vertices of the same side must either start or end with an edge in M and therefore that path cannot have only internal vertices in $V \setminus Z$. Thus, the torso operation does not add a shortcut edge between two vertices that are on the same side.

Now let G' be the underlying undirected graph corresponding to H' and let $k = |M[(V \setminus Z]| = |M| - |M_X|$, i.e., the number of edges of M outside of Z. We show that the maximum matching size of $G - (L' \cup R')$, for $L' \subseteq L_0$ and $R' \subseteq R_0$, can also be computed in G'.

▶ **Lemma 16.** *For every $L' \subseteq L_0$ and $R' \subseteq R_0$, the graph $G - (L' \cup R')$ has maximum matching size l if and only if $G' - (L' \cup R')$ has maximum matching size $l - k$.*

Proof. Let us fix $L' \subseteq L_0$ and $R' \subseteq R_0$ and denote $S = F_L \cup (L_0 \setminus L')$, $T = F_R \cup (R_0 \setminus R')$.

($\Rightarrow$) Let us first assume that $G - (L' \cup R')$ has a maximum matching M'_0 of size l. By Lemma 12 we have that there is a vertex-disjoint packing of M-augmenting paths $\mathcal{P}$ of size $l - |M|$ which we can use to augment M to a (maximum) matching M_0 of size $|M'_0|$. Note that M_0 agrees with M except for the augmenting paths in $\mathcal{P}$. (The same is not necessarily true for M'_0 since there could, e.g., be alternating cycles in $M'_0 \Delta M$.) Since every M-augmenting path corresponds to a directed path from S to T and $X \subseteq X' \subseteq Z$ contains a minimum (S, T)-cut we have that every path in $\mathcal{P}$ contains at least one vertex of Z. Furthermore $|\mathcal{P}| \leq |L_0| + |R_0|$ because every augmenting path must contain at least one vertex of $L_0 \cup R_0$ by M being maximum matching in $G - (L_0 \cup R_0)$ (see Observation 2).

We consider maximal subpaths of $\mathcal{P}$ with internal vertices not from Z and having at least one internal vertex. (Note that vertices in $(F_L \cup F_R) \setminus Z \subseteq F_L \cup F_R$ are M-unmatched, so paths in $\mathcal{P}$ cannot have such vertices as internal vertices: Those in F_L have no incoming edges and those in F_R have no outgoing edges.) If an internal vertex v on a path in $\mathcal{P}$ is in Z, then $v \in X' \subseteq Z$ since $Z \setminus X' \subseteq F_L \cup F_R$. If an internal vertex v of a path in $\mathcal{P}$ is in Z, and thus $v \in X'$, then we have that at least one neighbor of v on the path was also included in $X' \subseteq Z$ since v is incident with M (v is an internal vertex of an M-augmenting path). Therefore, these subpaths are vertex-disjoint. Let us consider each such subpath $P = (p, \ldots, q)$. We distinguish three cases, based on whether p and/or q are contained in Z, and apply a replacement operation to M_0 for each of them.

If $p, q \in Z$, then we have an edge $\{p, q\}$ in G' because it was either in the original graph, or the corresponding shortcut arc (p, q) was added to H' during the torso operation. Since no edge in M has exactly one vertex in Z, we have that P starts and ends with an edge of M_0, and there is exactly one less edge of M on P than there are edges of M_0 on P. We modify M_0 by removing all edges of M_0 on P, adding all edges of M on P, and adding $\{p, q\}$. After this modification the size of M_0 is the same as before and it is still a matching.

If $p \in Z$ and $q \notin Z$, then we have that $W_p \geq |L_0| + |R_0|$, since otherwise we would have $q \in F_v = W_v \subseteq Z$ since it is reachable from p with no internal vertices in $Z \supseteq X'$. We again have that the first and last edge of P start and end with an edge of M_0, since the first edge cannot be an edge of M because it has only one endpoint in Z, while the last edge is a final edge of a path in $\mathcal{P}$ by maximality of P and P ends in an M-unmatched vertex of F_R. (It cannot end in F_L because those vertices have no incoming edges.) We modify M_0 by removing all edges of M_0 on P, adding all edges of M on P and adding one edge from p into W_p. Because this case can occur at most $|L_0| + |R_0|$ times (at most once for every path in $\mathcal{P}$) we have that there is always a free vertex in W_p. Again, the size of M_0 remains the same.

We handle the case where $p \notin Z$ and $q \in Z$ similarly. Note that $p, q \notin Z$ cannot occur because then P would not a be maximal subpath with internal vertices not from Z since every path in $\mathcal{P}$ visits at least one vertex in Z. After handling every maximal subpath in this fashion there are no edges of M_0 with only one endpoint in Z. Furthermore, M_0 and M agree on all edges not incident to Z and thus $|M_0[V \setminus Z]| = |M[V \setminus Z]$. Hence, $M_0[Z]$, the restriction of M_0 to Z, is a matching in $G' - (L' \cup R')$ of size $|M_0[Z]| = |M_0| - |M_0[V \setminus Z]| = l - |M[V \setminus Z]| = l - k$.

($\Leftarrow$) Assume that $G' - (L' \cup R')$ has a matching M_0 of size $l - k$. Let $M' = M[Z] = M_X$ denote the restriction of M to Z. Since there are no edges of M with exactly one vertex in Z,

the set M' is a matching in $G' - (L' \cup R')$ of size $|M| - k$. By Lemma 12, there is a packing $\mathcal{P}$ of $r = (l-k) - |M'| = l - |M|$ vertex-disjoint M'-augmenting paths in $G' - (L' \cup R')$, which correspond to r vertex-disjoint directed paths from $S \cap Z$ to $T \cap Z$ in $H' - (L' \cup R')$.

By construction X contains a minimum (S,T)-cut Y in $H - (L' \cup R')$; suppose that $|Y| < r = |\mathcal{P}|$. There must be a path in $\mathcal{P}$ which avoids Y. This path corresponds to a directed path P from $S \cap Z$ to $T \cap Z$ in $H' - (L' \cup R')$. Arcs of P are either arcs in $H - (L' \cup R')$ or shortcut arcs which are added by the torso operation. Note that each shortcut arc corresponds to a directed path with no internal vertices from Z, which therefore also avoids $Y \subseteq X \subseteq Z$. Hence, if we replace shortcut arcs in P by corresponding paths in $H - (L' \cup R')$, then we obtain a directed walk from $S \cap Z$ to $T \cap Z$ in $H - (L' \cup R')$, which contradicts that Y is a (S,T)-cut in $H - (L' \cup R')$. Hence we have $|Y| \geq r$, which implies that there are at least r vertex-disjoint (S,T)-paths in $H - (L' \cup R')$. These paths correspond to M-augmenting paths in $G - (L' \cup R')$. Thus, $G - (L' \cup R')$ has a matching of size at least $|M| + r = |M| + (l - |M|) = l$. ◀

The size of X is polynomial $\mathcal{O}((|L_0| + |R_0|)^3)$. Therefore $|X'| = \mathcal{O}((|L_0| + |R_0|)^3)$ since we add at most $|X|$ more vertices that are an endpoint to a matched edge in M incident to X. To obtain Z, at most $|L_0| + |R_0|$ vertices are added for every vertex in X'. Thus, Z is of size $\mathcal{O}((|L_0| + |R_0|)^4)$ and therefore G' has at most $\mathcal{O}((|L_0| + |R_0|)^4)$ vertices. All operations required to obtain G' can be performed in polynomial time by using appropriate flow calculations. Correctness follows from Lemma 16, completing Theorem 10.

Uncertain edges. Our result can also be extended to the case when there are some *edges* of the input graph that may not be available; we call these edges *uncertain edges*. We proceed as follows: For each uncertain edge uv, we subdivide it into three edges $uu', u'v'$ and $v'v$ (note that this does not change the bipartiteness of the graph); then instead of considering uv as an uncertain edge, we consider u' and v' as uncertain vertices. The auxiliary vertices u' and v' can be used to control the availability of uv in the graph. We remove u' and v' to make uv unavailable and include both u and v in the final input to make uv available. Now we can repeat the algorithm in Theorem 10 with a small modification: Before applying the torso operation, we add u, u', v' and v to the cut-covering set. This ensures that the new edges and vertices that correspond to uncertain edges are not affected by the torso operation. After the torso operation has been applied, we obtain a new compressed graph with some uncertain vertices, and moreover all endpoints of uncertain edges and their subdivisions in the input graph are preserved. Finally, we may contract subdivided edges with auxiliary (uncertain) vertices to get the original uncertain edges.

The final thing we need to be careful about is how the size of maximum matching may change under subdivision (and contraction). Given a graph G, if we subdivide an edge uv of G into three edges $uu', u'v'$ and $v'v$ to obtain a graph G', then we increase the size of maximum matching of G by one. Indeed, we just need to see how to construct a matching in the new graph from a maximum matching M of G. If M contains uv then we just need to replace uv by uu' and vv'. Otherwise, we add $u'v'$ to M; in both cases the size increases by one, as claimed. Thus, we can conclude our result carries over to the case where for a subset of edges and vertices it is unknown if they appear in the final input.

6 Lower Bounds

In this section we present some unconditional lower bounds for preprocessing instances of some tractable problems in which part of the input is uncertain. We derive these lower

bounds from the Membership communication game which is defined as follows. We have two players, Alice and Bob. Alice has a subset $S \subseteq [n]$ and Bob has an integer l. The objective of the game is for Bob to find out if $l \in S$. It is well known that a one-way communication protocol from Alice to Bob has a cost of at least n bits of information. We show that for certain problems the existence of algorithms that are similar to those presented in this paper would give a more efficient protocol. We start with Connected Vertex Cover which is solvable in time $2^b|V|^{\mathcal{O}(1)}$ if we are looking for a solution of size at most b [3]. Therefore, it is solvable in polynomial time for $b \leq \log|V|$ in which case we refer to the problem as Small Connected Vertex Cover. The following theorem shows that we cannot find a succinct equivalent instance if for a set W of vertices it is uncertain if they appear in the final input.

▶ **Theorem 17.** *There is no algorithm that, given an instance $G = (V \cup W, E)$ of* Small Connected Vertex Cover*, outputs a graph G' of size $|W|^{\mathcal{O}(1)}$ and $k \in \mathbb{N}$ such that, for any $W' \subseteq W$, the graph $G - W'$ has a connected vertex cover of size at most $b \leq \log|V|$ if and only if $G' - W'$ has a connected vertex cover of size at most $b - k$.*

Note that this lower bound holds even if the time to compute G' and k is unbounded. Furthermore, we remark that by a similar information theoretic argument, any encoding from which we can extract the answer to Small Connected Vertex Cover for any $W' \subseteq W$ requires at least $2^{|W|/2} = n$ bits: Otherwise, since there are 2^n possible subsets of $[n]$, by the pigeonhole principle (we have fewer than 2^n distinct bit-strings of length smaller than n) there must be at least 2 subsets $S \neq S'$ of $[n]$ for which the encoding of the instance produced in the proof of Theorem 17 is the same. Now there must be an element i that is, w.l.o.g., in S but not in S' for which plugging in the corresponding W' produces the same answer, which is clearly wrong. We show a similar lower bound for Linear Programming where for part of the constraints it is uncertain if they appear in the final input.

▶ **Theorem 18.** *There is no algorithm that, given an LP with objective function $\boldsymbol{c}^T\boldsymbol{x}$ and constraints $A\boldsymbol{x} \leq \boldsymbol{b}, B\boldsymbol{x} \leq \boldsymbol{d}, \boldsymbol{x} \geq 0$, outputs an encoding of size smaller than $2^{c/2}$ from which we can correctly determine feasibility of the LP for any subset of constraints in $B\boldsymbol{x} \leq \boldsymbol{d}$, where c is the number of constraints in $B\boldsymbol{x} \leq \boldsymbol{d}$.*

Let us remark that a similar lower bound can be obtained for linear programming instances if for part of the constraints the right-hand side is unknown, and we would like to obtain an instance that is equivalent with respect to feasibility for any assignment of the right-hand side in these constraints. This follows from the proof of Theorem 18 when according to $s(i)$, we set $x_i \geq 1$ and $x_i \leq 1$ (resp. $x_i \geq 0$ and $x_i \leq 0$) if the i-th bit of $s(i)$ is set to 1 (resp. 0).

Finally, in a setting where the objective function of the LP is unknown we have a lower bound of 2^c where c is the number of variables: Alice uses the same constraints $Ax \leq b$ as in Theorem 18. Now, target functions $y_1 + \ldots + y_p$ can take value p if and only if there is no constraint $y_1 + \ldots + y_p \leq p - 0.5$, i.e., if and only if the corresponding element j is not in S.

7 Conclusion

We have initiated a programmatic study of preprocessing for efficiently solvable problems for the setting that not the entire input is known or that parts of the input are not certain, inspired by recent results [13, 8] obtained in the context of kernelization. Intuitively, incomplete or partially uncertain instances correspond to a possibly exponentially large family of similar instances. Thus, it is not clear (and as shown it does not hold in general) that one can efficiently generate an instance of size polynomial in the amount of uncertain data that

allows to compute optimal solutions for the initial input for each instantiation (and without access to the initial input of course). This direction of research seems to apply for a variety of polynomial-time solvable problems (and of course also for NP-hard ones). In particular, different notions of missing or uncertain data may be suitable for the same problem. We have considered weight functions whose values are unknown for certain items as well as graphs in which some vertices or edges may or may not appear in the instantiation.

Natural problems for future research are for example matchings in general graphs (with uncertain vertices, edges, and or weights) and MAX FLOW with arbitrary and uncertain capacities. It would also be interesting whether there are particular lower bound techniques for this form of preprocessing, e.g., to prove that a certain amount of information is optimal.

References

1. Sepehr Assadi, Sanjeev Khanna, Yang Li, and Val Tannen. Dynamic sketching for graph optimization problems with applications to cut-preserving sketches. *CoRR*, abs/1510.03252, 2015. URL: http://arxiv.org/abs/1510.03252.
2. Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011. doi:10.1137/080734510.
3. Marek Cygan. Deterministic parameterized connected vertex cover. In *SWAT 2012*, volume 7357 of *LNCS*, pages 95–106. Springer, 2012. doi:10.1007/978-3-642-31155-0_9.
4. George B. Dantzig. Linear programming under uncertainty. *Management Science*, 50(12-Supplement):1764–1769, 2004. doi:10.1287/mnsc.1040.0261.
5. Reinhard Diestel. *Graph theory (Graduate texts in mathematics)*. Springer Heidelberg, 2005.
6. Stefan Fafianie, Stefan Kratsch, and Vuong Anh Quyen. Preprocessing under uncertainty. *CoRR*, abs/1510.05503, 2015. URL: http://arxiv.org/abs/1510.05503.
7. Michael Hoffmann, Thomas Erlebach, Danny Krizanc, Matús Mihalák, and Rajeev Raman. Computing minimum spanning trees with uncertainty. In *STACS 2008*, volume 1 of *LIPIcs*, pages 277–288, 2008. doi:10.4230/LIPIcs.STACS.2008.1358.
8. Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *FOCS 2012*, pages 450–459. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.46.
9. Dániel Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci.*, 410(44):4471–4479, 2009. doi:10.1016/j.tcs.2009.07.027.
10. Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms*, 9(4):30, 2013. doi:10.1145/2500119.
11. Nicole Megow, Julie Meißner, and Martin Skutella. Randomization helps computing a minimum spanning tree under uncertainty. In *ESA 2015*, volume 9294 of *LNCS*, pages 878–890. Springer, 2015. doi:10.1007/978-3-662-48350-3_73.
12. James Oxley. *Matroid Theory*. Oxford University Press, 2011.
13. Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network sparsification for Steiner problems on planar and bounded-genus graphs. In *FOCS 2014*, pages 276–285. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.37.