Journal of Computer and System Sciences 92 (2018) 48–64

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

On the complexity of trial and error for constraint satisfaction problems ☆

Gábor Ivanyos [a], Raghav Kulkarni [b], Youming Qiao [c,*], Miklos Santha [b,d], Aarthi Sundaram [b]

[a] Institute for Computer Science and Control, Hungarian Academy of Sciences, Budapest, Hungary
[b] Centre for Quantum Technologies, National University of Singapore, Singapore 117543, Singapore
[c] Centre or Quantum Software and Information, University of Technology Sydney, Australia
[d] IRIF, CNRS, Université Paris Diderot, 75205 Paris, France

ARTICLE INFO

Article history:
Received 24 October 2016
Received in revised form 4 June 2017
Accepted 21 July 2017

Keywords:
Trial and error
Constraint satisfaction problems
Computational complexity

ABSTRACT

In 2013 Bei, Chen and Zhang introduced a trial and error model of computing, and applied to some constraint satisfaction problems. In this model the input is hidden by an oracle which, for a candidate assignment, reveals some information about a violated constraint if the assignment is not satisfying. In this paper we initiate a *systematic* study of constraint satisfaction problems in the trial and error model, by adopting a formal framework for CSPs, and defining several types of revealing oracles. Our main contribution is to develop a *transfer theorem* for each type of the revealing oracle. To any hidden CSP with a specific type of revealing oracle, the transfer theorem associates another CSP in the normal setting, such that their complexities are polynomial-time equivalent. This in principle transfers the study of a large class of hidden CSPs to the study of normal CSPs. We apply the transfer theorems to get polynomial-time algorithms or hardness results for several families of concrete problems.

1. Introduction

In [3], Bei, Chen and Zhang proposed a *trial and error* model to study algorithmic problems when some input information is lacking. As argued in their paper, the lack of input information can happen when we have only limited knowledge of, and access to the problem. They also described several realistic scenarios where the inputs are actually unknown. Then, they formalized this methodology in the complexity-theoretic setting, and proposed a trial and error model for constraint satisfaction problems. They further applied this idea to investigate the information needed to solve linear programming in [4], and to study information diffusion in a social network in [1].

As mentioned, in [3] the authors focused on the hidden versions of some specific constraint satisfaction problems (H–CSPs), whose instances could only be accessed via a *revealing* oracle. An algorithm in this setting interacts with this revealing oracle to get information about the input instance. Each time, the algorithm proposes a candidate solution, a *trial*, and the validity of this trial is checked by the oracle. If the trial succeeds, the algorithm is notified that the proposed trial is already a solution. Otherwise, the algorithm obtains as an *error*, a violation of some property corresponding to the instance.

☆ A preliminary report on this work appeared in ICALP 14 as [7].

* Corresponding author.
E-mail addresses: Gabor.Ivanyos@sztaki.mta.hu (G. Ivanyos), kulraghav@gmail.com (R. Kulkarni), jimmyqiao86@gmail.com (Y. Qiao), santha@irif.fr (M. Santha), aarthims@u.nus.edu (A. Sundaram).

http://dx.doi.org/10.1016/j.jcss.2017.07.005
0022-0000/

The algorithm aims to make effective use of these errors to propose new trials, and the goal is to minimize the number of trials while keeping in mind the cost for proposing new trials. When the CSP is already difficult, a computation oracle that solves the original problem might be allowed. Its use is justified as we are interested in the *extra difficulty* caused by the lack of information. Bei, Chen and Zhang considered several natural CSPs in the trial and error setting, including SAT, Stable Matching, Graph Isomorphism and Group Isomorphism. While the former two problems in the hidden setting are shown to be of the same difficulty as in the normal one, the last two cases have substantially increased complexities in the unknown-input model. They also studied more problems, as well as various aspects of this model, like the query complexity.

In this paper, following [3], we initiate a *systematic* study of the constraint satisfaction problems in the trial and error model. To achieve this, we first adopt a formal framework for CSPs, and based on this framework we define three types of revealing oracles. This framework also helps to clarify and enrich the model of [3]. Let us make a quick remark that, our CSP model has a couple of features that may not be quite standard. We will mention some of these in the following, and discuss these in detail in Section 2.3.

Our main contribution is to develop a *transfer theorem* for each type of the revealing oracle, under a broad class of parameters. For any hidden CSP with a specific type of revealing oracle, the transfer theorem associates another CSP in the normal (unhidden) setting, such that their difficulties are roughly the same. This in principle transfers the study of hidden CSPs to the study of CSPs in the normal setting. We also apply transfer theorems to get results for concrete CSPs, including some problems considered in [3], for which we usually get much shorter and easier proofs.

The framework for CSPs*, and hidden* CSPs To state our results we describe informally the framework of CSPs. A CSP S is defined by a finite alphabet $[\![w]\!] = \{0, 1, \ldots, w-1\}$ and by $\mathcal{R} = \{R_1, \ldots, R_s\}$, a set of relations over $[\![w]\!]$ of some fixed arity q. For a set of variables $\mathcal{V} = \{x_1, \ldots, x_\ell\}$, an instance of S is a set of constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$, where $C_j = R(x_{j_1}, \ldots, x_{j_q})$ for some relation $R \in \mathcal{R}$ and some q-tuple of variables. An assignment $a \in [\![w]\!]^\ell$ satisfies $\mathcal{C}$ if it satisfies every constraint in it.

Example 1.1. 1SAT: Here $w = 2$, $q = 1$, and $\mathcal{R} = \{\mathsf{Id}, \mathsf{Neg}\}$, where $\mathsf{Id} = \{1\}$ is the identity relation, and $\mathsf{Neg} = \{0\}$ is its complement. Thus a constraint is a literal x_i or $\bar{x}_i$, and an instance is just a collection of literals. In case of 3SAT the parameters are $w = 2$, $q = 3$ and $|\mathcal{R}| = 8$. We will keep for further illustrations 1SAT which is a problem in polynomial time. 3SAT would be a less illustrative example since the standard problem is already NP-complete.

To allow for more versatility, we may only be interested in those assignments satisfying certain additional conditions that cannot be (easily) expressed in the framework of constraint satisfaction problems. This case happens, say when we look for permutations in isomorphism problems, or when we view monotone graph properties as CSP problems in Section 6. To cover these cases, our model will also include a subset $W \subseteq [\![w]\!]^\ell$ as a parameter and we will look for satisfying assignments from W, whose elements will be refereed to as *admissible assignments*. That these admissible assignments can play a notable role (as in Section 6) is a first feature that may be somewhat surprising.

Recall that in the hidden setting, the algorithm interacts with some revealing oracle by repeatedly proposing assignments. If the proposed assignment is not satisfying then the revealing oracle discloses certain information about some violated constraint. This can be in principle an index of such a constraint, (the index of) the relation in it, the indices of the variables where this relation is applied, or any subset of the above. Here we will require that the oracle always reveals the index of a violated constraint from $\mathcal{C}$. To characterize the choices for the additional information, for any subset $\mathcal{U} \subseteq \{\mathcal{R}, \mathcal{V}\}$ we say that an oracle is $\mathcal{U}$*-revealing* if it also gives out the information corresponding to $\mathcal{U}$. For a CSP problem S we use H–$\mathrm{S}_{\mathcal{U}}$ to denote the corresponding hidden problem in the trial and error model with $\mathcal{U}$-revealing oracle.

Example 1.1 *(continued).* Let us suppose that we present an assignment $a \in \{0,1\}^\ell$ for an instance of the hidden version H–$\mathrm{1SAT}_{\mathcal{U}}$ of 1SAT to the $\mathcal{U}$-revealing oracle. If $\mathcal{U} = \{\mathcal{V}\}$ and the oracle reveals j and i respectively for the violated constraint and the variable in it then we learn that the jth literal is x_i if $a_i = 0$, and $\bar{x}_i$ otherwise. If $\mathcal{U} = \{\mathcal{R}\}$ and say the oracle reveals j and Id then we learn that the jth literal is positive. If $\mathcal{U} = \emptyset$ and the oracle reveals j then we only learn that the jth literal is either a positive literal corresponding to one of the indices where a is 0, or a negative literal corresponding to an index where a is 1.

In order to explain the transfer theorem and motivate the operations which create richer CSPs, we first make a simple observation that H–$\mathrm{S}_{\{\mathcal{R},\mathcal{V}\}}$ and S are polynomial time equivalent, when the relations of S are in P. Indeed, an algorithm for H–$\mathrm{S}_{\{\mathcal{R},\mathcal{V}\}}$ can solve S, as the answers of the oracle can be given by directly checking if the proposed assignment is satisfying. In the other direction, we repeatedly submit assignments to the oracle. The answer of the oracle fully reveals a (violated) constraint. Given some subset of constraints we already know, to find a new constraint, we submit an assignment which satisfies all the known constraints. Such an assignment can be found by the algorithm for S.

With a weaker oracle this procedure clearly does not work and to compensate, we need stronger CSPs. In the case of $\{\mathcal{V}\}$-revealing oracles an answer helps us exclude, for the specified clause, all those relations which were satisfied at the specified indices of the proposed assignment, but keep as possibilities all the relations which were violated at those indices. Therefore, to find out more information about the input, we would like to find a satisfying assignment for a CSP instance

whose corresponding constraint is the union of the violated relations. This naturally brings us to consider the constraint satisfaction problem $\bigcup$S, the *closure by union* of S. The relations for $\bigcup$S are from $\bigcup\mathcal{R}$, the *closure by union* of $\mathcal{R}$, which contains relations by taking union over any subset of $\mathcal{R}$.

The situation with the $\{\mathcal{R}\}$-revealing oracle is analogous, but here we have to compensate, in the stronger CSP, for the lack of revealed information about the variable indices. For a relation R and q-tuple of indices $(j_1, \dots, j_q)$, we define the ℓ-ary relation $R^{(j_1,\dots,j_q)} = \{a \in W : (a_{j_1}, \dots, a_{j_q}) \in R\}$, and for a set I of q-tuples of indices, we set $R^I = \bigcup_{(j_1,\dots,j_q)\in I} R^{(j_1,\dots,j_q)}$. The *arity extension* of S is the constraint satisfaction problem E–S whose relations are from arity extension E–$\mathcal{R} = \bigcup_I \{R^I : R \in \mathcal{R}\}$ of $\mathcal{R}$. Note that the arity extension produces relations whose arities are as large as the assignment length. This requires us to consider CSPs where the arities of relations can be functions in e.g. the assignment length. While such CSPs include some natural instances like systems of linear equalities, this feature may also be unfamiliar to some readers.

The transfer theorem first says that with $\bigcup$S (resp. E–S) we can compensate the information hidden by a $\{\mathcal{V}\}$-revealing (resp. $\{\mathcal{R}\}$-revealing) oracle, that is we can solve H–S$_{\{\mathcal{V}\}}$ (resp. H–S$_{\{\mathcal{R}\}}$). In fact, with $\bigcup$E–S we can solve H–S$_\emptyset$. Moreover, perhaps more surprisingly, it says that these statements also hold in the reverse direction: if we can solve the hidden CSP, we can also solve the corresponding extended CSP.

Transfer Theorem *(informal statement). Let* S *be a CSP whose parameters are "reasonable" and whose relations are in* P. *Then for any promise* W *on the assignments, the complexities of the following problems are polynomial time equivalent: (a)* H–S$_{\{\mathcal{V}\}}$ *and* $\bigcup$S, *(b)* H–S$_{\{\mathcal{R}\}}$ *and* E–S, *(c)* H–S$_\emptyset$ *and* $\bigcup$E–S.

The precise dependence on the parameters can be found in the theorems of Section 3. Corollary 3.4 highlights the conditions for polynomial equivalence.

Example 1.1 *(continued).* Since $\bigcup\{\mathsf{Id}, \mathsf{Neg}\} = \{\emptyset, \mathsf{Id}, \mathsf{Neg}, \{0,1\}\}$, $\bigcup$1SAT has only the two trivial (always false or always true) relations in addition to the relations in 1SAT. Therefore it can be solved in polynomial time, and by the Transfer Theorem H–1SAT$_{\{\mathcal{V}\}}$ is also in P. On the other hand, for any index set $I \subseteq [\ell]$, Id^I is a disjunct of positive literals with variables from I, and similarly Neg^I is a disjunct of negative literals with variables from I. Thus E–1SAT includes MONSAT, which consists of those instances of SAT where in each clause either every variable is positive, or every variable is negated. The problem MONSAT is NP-hard by Schaefer's characterization [10], and therefore the Transfer Theorem implies that H–1SAT$_{\{\mathcal{R}\}}$ and H–1SAT$_\emptyset$ are also NP-hard.

In a further generalization, we will also consider CSPs and H–CSPs whose instances satisfy some property. One such property can be *repetition freeness* meaning that the constraints of an instance are pairwise distinct. The promise H–CSPs could also be a suitable framework for discussing certain graph problems on special classes of graphs. For a promise PROM on instances of S we denote by S$^{\mathsf{PROM}}$ the promise problem whose instances are instances of S satisfying PROM. The problem H–S$^{\mathsf{PROM}}_{\{\mathcal{U}\}}$ is defined in an analogous way from H–S$_{\{\mathcal{U}\}}$.

It turns out that we can generalize the Transfer Theorem for CSPs with promises on the instances. We describe this in broad lines for the case of $\{\mathcal{V}\}$-revealing oracles. Given a promise PROM on S, the corresponding promise $\bigcup$PROM for $\bigcup$S is defined in a natural way. We say that a $\bigcup$S-instance $\mathcal{C}'$ *includes* an S-instance $\mathcal{C}$ if for every $j \in [m]$, the constraint C'_j in $\mathcal{C}'$ and the constraint C_j in $\mathcal{C}$ are defined on the same variables, and seen as relations, $C_j \subseteq C'_j$. Then $\bigcup$PROM is the set of instances $\mathcal{C}'$ of $\bigcup$S which include some $\mathcal{C} \in$ PROM. The concept of an algorithm *solving* $\bigcup$S$^{\bigcup\mathsf{PROM}}$ has to be relaxed: while we search for a satisfying assignment for those instances which include a satisfiable instance of PROM, when this is not the case, the algorithm can abort even if the instance is satisfiable. With this we have:

Transfer Theorem for promise problems *(informal statement). Let* S *be a constraint satisfaction problem with promise* PROM. *Then the complexities of* H–S$^{\mathsf{PROM}}_{\{\mathcal{V}\}}$ *and* $\bigcup$S$^{\bigcup\mathsf{PROM}}$ *are polynomial time equivalent when the parameters are "reasonable" and the relations of* S *are in* P.

Example 1.1 *(continued).* Let RF denote the property of being repetition free, in the case of 1SAT this just means that no literal can appear twice in the formula. Then H–1SAT$^{\mathsf{RF}}_\emptyset$, hidden repetition-free 1SAT with $\emptyset$-revealing oracle, is solved in polynomial time. To see this we first consider X–1SAT, the constraint satisfaction problem whose relations are all ℓ-ary extensions of Id and Neg. (See Section 2 for a formal definition.) It is quite easy to see that hidden 1SAT with $\emptyset$-revealing oracle is essentially the same problem as hidden X–1SAT with $\{\mathcal{V}\}$-revealing oracle. Therefore, by the Transfer Theorem we are concerned with $\bigcup$X–1SAT with promise $\bigcup$RF. The instances satisfying the promise are $\{C_1, \dots, C_m\}$, where C_j is a disjunction of literals such that there exist distinct literals $z_1, \dots, z_m$, with $z_j \in C_j$. It turns out that these specific instances of SAT can be solved in polynomial time. The basic idea is that we can apply a maximum matching algorithm, and only output a solution if we can select m pairwise different variables $x_{i_1}, \dots, x_{i_m}$ such that either x_{i_j} or $\overline{x}_{i_j}$ is in C_j.

Applications of transfer theorems. Since NP-hard problems obviously remain NP-hard in the hidden setting (without access to an NP oracle), we investigate the complexity of various polynomial-time solvable CSPs. We first apply the Transfer

Theorem when there is no promise on the instances. We categorize the hidden CSPs depending on the type of the revealing oracle.

With constraint and variable index revealing oracles, we obtain results on several interesting families of CSPs including the exact-Unique Games Problem (cf. Section 4), equality to a member of a fixed class of graphs. Interestingly, certain CSPs, like 2SAT and the exact-Unique Game problem on alphabet size 2 remain in P, while some other CSPs like the exact-Unique Game problem on alphabet size ≥ 3, and equality to some specific graph, such as k-cliques, become NP-hard in this hidden input setting. The latter problem is just the Graph Isomorphism problem considered in [2, Theorem 13], whose proof, with the help of the Transfer Theorem, becomes very simple.

With constraint and relation index revealing oracles, we show that if the arity and the alphabet size are constant, any CSP satisfying certain modest requirement becomes NP-hard. To be specific, we require that that for every element α of the alphabet, the collection $\mathcal{R}$ contains a relation which is *violated* by the tuple $(\alpha, \ldots, \alpha)$. This can be justified by observing that otherwise it would be easy to find a satisfying assignment for any instance using $O(w)$ trials.

We then study various monotone graph properties like Spanning Tree, Cycle Cover, etc. We define a general framework to represent variants of monotone graph property problems as H–CSPs. Since in this framework only one relation is present, the relation index is not a concern. We deal with the constraint index revealing oracle, which is equivalent to the constraint and relation index revealing oracle in this case, and prove that the problems become NP-hard. This framework also naturally extends to directed graphs.

Finally, we investigate hidden CSPs with promises on the instances. We first consider the repetition freeness promise, as exhibited by the 1SAT example as above. Though the hidden repetition free 1SAT problem becomes solvable in polynomial time, 2SAT is still NP-hard. The group isomorphism problem can also be cast in this framework, and we give a simplified proof of [3, Theorem 11]: to compute an explicit isomorphism of the hidden group with $\mathbb{Z}_p$ is NP-hard.

Comparisons with [3] We now compare our framework and results with those in [3] explicitly. Recall that we defined three revealing oracles, $\emptyset$-, $\{\mathcal{V}\}$-, and $\{\mathcal{R}\}$-revealing oracles. The $\emptyset$-revealing oracle was the original setting discussed in [3]. The $\{\mathcal{V}\}$- and $\{\mathcal{R}\}$-revealing oracles are new, so are the results about specific CSPs in the setting of these two oracles. For the $\emptyset$-revealing oracle, both [3] and this paper discussed SAT and Group Isomorphism. Isomorphisms of two graphs and isomorphisms of a graph with a clique were studied in the report [2]. Here we prove a hardness result for the latter problem. The paper [3] further considered several other problems including stable matching, and Nash Equilibrium. On the other hand the monotone graph properties (Section 6) and certain promise problems (Section 7) are studied only in this paper.

Bei, Chen and Zhang also gave bounds on the *trial complexity* of some of the problems considered in [3], including stable matching and SAT. (The trial complexity measures the number of oracle calls to solve them in the hidden model). Although the algorithms outlined in the proofs for our transfer theorems provide generic upper bounds on the trial complexity, giving tighter bounds would be beyond the focus of the present paper.

Organization. In Section 2 we formally describe the model of CSPs, and hidden CSPs. In Section 3, the transfer theorems are stated and proved. Section 4, 5, and 6 contain the applications of the main theorems in the case of $\{\mathcal{V}\}$-revealing oracle, $\{\mathcal{R}\}$-revealing oracle, and monotone graph properties, respectively. Finally in Section 7 we present the results for hidden promise CSPs.

2. Preliminaries

2.1. The model of constraint satisfaction problems

For a positive integer k, let $[k]$ denote the set $\{1, \ldots, k\}$, and let $[\![k]\!] = \{0, 1, \ldots, k-1\}$. A *constraint satisfaction problem*, (CSP) S, is specified by its set of parameters and its type, both defined for every positive integer n.

The parameters The *parameters* are the alphabet size $w(n)$, the assignment length $\ell(n)$, the set of (admissible) assignments $W(n) \subseteq [\![w(n)]\!]^{\ell(n)}$, the arity $q(n)$, and the number of relations $s(n)$. To simplify notations, we often omit n from the parameters, and just write w, ℓ, W, q and s.

We suppose that the parameters, as functions of n, can be computed in time polynomial in n. In many cases (like in classical CSPs) n coincides with ℓ, the assignment length but for e.g. (monotone) graph properties the n is the number of vertices while the assignment length is $\binom{n}{2}$, the number of possible edges.

The type For a sequence $J = (j_1, \ldots, j_q)$ of q distinct indices we denote W_J the projection of W to the coordinates from J: $W_J = \{(v_1, \ldots, v_q) \in [w]^q : \exists (w_1, \ldots, w_\ell) \in W$ with $w_{j_i} = v_i\}$. We suppose that W_J does not depend on the choice of J, that is, for every J consisting of q distinct indices we have $W_J = W_q := \{u \in [\![w]\!]^q : uv \in W$ for some $v \in [\![w]\!]^{\ell-q}\}$. This condition holds trivially for most cases, and for other cases (e.g. CSPs related to graphs), holds due certain symmetry condition there (e.g. graph properties are invariant for isomorphic graphs). A q-ary *relation* is $R \subseteq W_q$. For b in W_q, if $b \in R$, we sometimes write $R(b) = \mathrm{T}$, and similarly for $b \notin R$ we write $R(b) = \mathrm{F}$. The *type* of S is a set of q-ary relations $\mathcal{R}_n = \{R_1, \ldots, R_s\}$, where $R_k \subseteq W_q$, for every $k \in [s]$. As for the parameters, we usually just write $\mathcal{R}$. Observe that the type of a CSP automatically defines among its parameters the arity and the number of relations.

We assume that the alphabet set and the relation set have succinct representations. Specifically, every letter and relation can be encoded by strings over $\{0,1\}$ of length polynomial in n, and given such a string, we can decide whether it is a valid letter or relation efficiently. We also suppose the existence of Turing machines that, given n, words $b \in [\![w]\!]^q$, $v \in [\![w]\!]^\ell$, $k \in [s]$, decide whether $v \in W$, whether $b \in W_q$ and compute $R_k(b)$ if $b \in W_q$. We further introduce the following notations. For a relation R let $\mathrm{comp}(R)$ be the time complexity of deciding the membership of a tuple in R, and for a set of relations $\mathcal{R}$ let $\mathrm{comp}(\mathcal{R})$ be $\max_{R\in\mathcal{R}} \mathrm{comp}(R)$. We denote by $\dim(\mathcal{R})$, the *dimension* of $\mathcal{R}$, which is the maximum of the integers d such that there exists $R_1, \ldots, R_d \in \mathcal{R}$ with $R_1 \subsetneq R_2 \subsetneq \ldots \subsetneq R_d$. In other words, $\dim(\mathcal{R})$ is the length of the longest chain of relations (for inclusion) in $\mathcal{R}$.

The instances We set $[\ell]^{(q)} = \{(j_1, \ldots, j_q) \in [\ell]^q : |\{j_1, \ldots, j_q\}| = q\}$, that is $[\ell]^{(q)}$ denotes the set of distinct q-tuples from $[\ell]$. An *instance* of S is given by a set of m constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$ over a set $\mathcal{V} = \{x_1, \ldots, x_\ell\}$ of variables, where the *constraint* C_j is $R_{k_j}(x_{j_1}, \ldots, x_{j_q})$ for some $k_j \in [s]$ and $(j_1, \ldots, j_q) \in [\ell]^{(q)}$. We say that an assignment $a \in W$ satisfies $C_j = R_{k_j}(x_{j_1}, \ldots, x_{j_q})$ if $R_{k_j}(a_{j_1}, \ldots, a_{j_q}) = \mathrm{T}$. An assignment *satisfies* $\mathcal{C}$ if it satisfies all its constraints. The *size* of an instance is $n + m(\log s + q \log \ell) + \ell \log w$ which includes the length of the description of $\mathcal{C}$ and the length of the assignments. In all our applications the instance size will be polynomial in n. A *solution* of $\mathcal{C}$ is a satisfying assignment if there exists any, and NO otherwise.

Note that the size of an instance does not count the descriptions of the relations and the admissible assignments. This is because the latter information is thought of as the meta data of a CSP, and is known to the algorithm.

Operations creating new CSPs *from old* CSPs We also introduce two new operations which create richer sets of relations from a relation set. For a given CSP S, these richer sets of relations derived from the type of S, will be the types of harder CSPs which turn out to be equivalent to various hidden variants of S. The first operation is standard. We denote by $\bigcup\mathcal{R}$ the closure of $\mathcal{R}$ by the union operation, that is $\bigcup\mathcal{R} = \{\bigcup_{R\in\mathcal{R}'} R : \mathcal{R}' \subseteq \mathcal{R}\}$. We define the (*closure by*) *union* of S as the constraint satisfaction problem $\bigcup$S whose type is $\bigcup\mathcal{R}$, and whose other parameters are the same as those of S. We assume that a relation R in $\bigcup\mathcal{R}$ is represented a list of indices for relations from $\mathcal{R}$ whose union is R. We remark that $\dim(\bigcup\mathcal{R}) \le |\mathcal{R}|$.

For a relation $R \in \mathcal{R}$ and for $(j_1, \ldots, j_q) \in [\ell]^{(q)}$, we define the ℓ-ary relation $R^{(j_1,\ldots,j_q)} = \{a \in W : (a_{j_1}, \ldots, a_{j_q}) \in R\}$, and the *arity extension* of $\mathcal{R}$, as $\mathrm{X}\text{–}\mathcal{R} = \{R^{(j_1,\ldots,j_q)} : R \in \mathcal{R} \text{ and } (j_1, \ldots, j_q) \in [\ell]^{(q)}\}$. The set $\mathrm{X}\text{–}\mathcal{R}$ contains the natural extensions of relations in $\mathcal{R}$ to ℓ-ary relations, where the extensions of the same relation are distinguished according to the choice of the q-tuple where the assignment is evaluated. The *arity extension* of S is the constraint satisfaction problem X–S whose type is $\mathrm{X}\text{–}\mathcal{R}$, and whose other parameters are otherwise the same as those of S. We assume that a relation R in $\mathrm{X}\text{–}\mathcal{R}$ is represented by the index of a relation in $\mathcal{R}$ and a sequence from $[\ell]^{(q)}$.

The combination of these two operations applied to $\mathcal{R}$ gives $\bigcup\,\mathrm{X}\text{–}\mathcal{R}$, the union of the arity extension of $\mathcal{R}$, which contains arbitrary unions of arity extended relations. It will be useful to consider also restricting the unions to extensions coming from the same base relation. For $I \subseteq [\ell]^{(q)}$, we set $R^I = \bigcup_{(j_1,\ldots,j_q)\in I} R^{(j_1,\ldots,j_q)}$, and we define $\mathrm{E}\text{–}\mathcal{R} = \{R^I : R \in \mathcal{R} \text{ and } I \subseteq [\ell]^{(q)}\}$. Relations from $\bigcup\,\mathrm{X}\text{–}\mathcal{R}$ are assumed to be represented as a sequence of pairs, each consisting of an index of a relation in $\mathcal{R}$ and an element of $[\ell]^{(q)}$.

The *restricted union of arity extension* of S is the constraint satisfaction problem E–S whose type is $\mathrm{E}\text{–}\mathcal{R}$, and whose other parameters are otherwise the same as those of S. Observe that $\mathrm{E}\text{–}\mathcal{R} \subseteq \bigcup\mathrm{X}\text{–}\mathcal{R} = \bigcup\mathrm{E}\text{–}\mathcal{R}$.

2.2. *Hidden* CSP *in the trial and error model*

Suppose that we want to solve a CSP problem S whose parameters and type are known to us, but for the instance $\mathcal{C}$, we are explicitly given only n and the number of constraints m. The instance is otherwise specified by a *revealing* oracle V for $\mathcal{C}$ which can be used by an algorithm to receive information about the constraints in $\mathcal{C}$. The algorithm can propose $a \in W$ to the oracle which is conceived as its guess for a satisfying assignment. If a indeed satisfies $\mathcal{C}$ then V answers YES. Otherwise there exists some violated constraint $C_j = R_{k_j}(x_{j_1}, \ldots, x_{j_q})$, and the oracle has to reveal some information about that. We will require that the oracle always reveals j, the index of the constraint C_j in $\mathcal{C}$, but in addition, it can also make further disclosures. It can be the relation R_{k_j} in $\mathcal{R}$ (or equivalently its index k_j); it can also be $(j_1, \ldots, j_q)$, the q-tuple of indices of the ordered variables $x_{j_1}, \ldots, x_{j_q}$ in $\mathcal{V}$; or both of these. To characterize the choices for the additional information, for any subset $\mathcal{U} \subseteq \{\mathcal{R}, \mathcal{V}\}$, we require that a *$\mathcal{U}$-revealing* oracle $\mathsf{V}_\mathcal{U}$ give out the information corresponding to $\{\mathcal{C}\}\bigcup\mathcal{U} \subseteq \{\mathcal{C}, \mathcal{R}, \mathcal{V}\}$. Thus for example a $\emptyset$-revealing oracle $\mathsf{V}_\emptyset$ reveals the index j of some violated constraint but nothing else, whereas a $\mathcal{V}$-revealing oracle $\mathsf{V}_{\{\mathcal{V}\}}$ also reveals the indices $(j_1, \ldots, j_q)$ of the variables of the relation in the clause C_j, but not the name of the relation.

Analogously, for every CSP S, and for every $\mathcal{U} \subseteq \{\mathcal{R}, \mathcal{V}\}$, we define the *hidden constraint satisfaction problem* (H–CSP) with *$\mathcal{U}$-revealing oracle* $\mathrm{H}\text{–}\mathrm{S}_\mathcal{U}$ whose parameters and type are those of S, but whose instances are specified by a $\mathcal{U}$-revealing oracle. An algorithm *solves* the problem $\mathrm{H}\text{–}\mathrm{S}_\mathcal{U}$ if for all n, m, for every instance $\mathcal{C}$ for S, specified by any $\mathcal{U}$-revealing oracle for $\mathcal{C}$, it outputs a satisfying assignment if there exists any, and NO otherwise. The complexity of an algorithm for $\mathrm{H}\text{–}\mathrm{S}_\mathcal{U}$ is the number of steps in the worst case over all inputs and all $\mathcal{U}$-revealing oracles, where a query to the oracle is counted as one step.

2.3. Discussion on the model

Our CSP model as described in Section 2.1 includes the usual model of CSPs, with a few additional features. Recall that we defined the alphabet size $w(n)$, the arity $q(n)$, and the number of relations $s(n)$ to be functions in n.[1] The most common case is of course when these functions are just constant. We allow them to be functions for the following two reasons. Firstly, this allows us to include a couple of natural problems, like systems of linear equations over a finite field ($q(n)$ and $s(n)$ are not constant; see Claim 5.2) and hyperplane non-cover ($w(n)$ is not constant; see Section 4). Secondly, and more importantly, the arity extension produces relations whose arities q are the same as the input length ℓ, which then depends on n. This also makes the number of relations dependent on n, and the union operator may introduce even more new relations.

We see from above that, allowing w, q, and s to be functions in n is not only flexible, but also necessary for our purposes. This may cause some problems though, if we do not pose any constraint on such functions. We remedy these as follows, as already described in Section 2.1. Firstly, we assume that all these functions can be computed in time polynomial in n. Secondly, we assume that the alphabet set and the relation set have succinct representations, and that membership of tuples in every relation R can be decided in time $\mathrm{comp}(R)$.

While the above measures may look somewhat inconvenient, in all concrete CSPs considered in this paper, they are satisfied in a straightforward manner. In most cases, $w(n)$ is polynomial in n; the only exception is hyperplane non-cover in Section 4. The non-constant arity situation is mostly caused by arity extensions, in which case the new arity is just the assignment length ℓ, and the number of new relations can be computed efficiently easily. The only problem with non-constant arities, not caused by arity extensions, is systems of linear questions as studied in Claim 5.2. The relations created by the union operation or the arity extension operation have natural succinct representations, and the number of such relations can be easily computed. Furthermore, starting with a set of relations $\mathcal{R}$ with $\mathrm{comp}(\mathcal{R})$, whether an admissible assignment $v \in W$ satisfies a relation in $\bigcup$ X–$\mathcal{R}$ involving $I \subseteq [\ell]^{(q)}$ can be computed in time $O(|I| \cdot |\mathcal{R}| \cdot \mathrm{comp}(\mathcal{R}))$.

Last but not least, the set of admissible assignments W can also play a crucial role. For example in Section 6 we discuss monotone graph properties and the admissible assignments are minimal graphs satisfying a particular graph property. This set W may cause similar problems if we do not pose any conditions on it, so we require that the membership of W can be computed. For all cases in this paper this is satisfied trivially.

3. Transfer theorems for hidden CSPs

In this section we state and prove our transfer theorems between H–CSPs and CSPs with extended types.

Theorem 3.1. *(a) If* $\bigcup$S *is solvable in time* T *then* H–$\mathrm{S}_{\{\mathcal{V}\}}$ *is solvable in time* $O((T + s \times \mathrm{comp}(\mathcal{R})) \times m \times \min\{\dim(\bigcup \mathcal{R}), |W_q|\})$.
(b) If H–$\mathrm{S}_{\{\mathcal{V}\}}$ *is solvable in time* T *then* $\bigcup$S *is solvable in time* $O(T \times m \times \mathrm{comp}(\bigcup \mathcal{R}))$.

We stress that in the theorem above instances of H–$\mathrm{S}_{\{\mathcal{V}\}}$ consisting of m relations correspond to instances of $\bigcup$S also consisting of m relations.

Proof. We first prove (a). Let $\mathcal{A}$ be an algorithm which solves $\bigcup$S in time T. We define an algorithm $\mathcal{B}$ for H–$\mathrm{S}_{\{\mathcal{V}\}}$. The algorithm will repeatedly call $\mathcal{A}$, until it finds a satisfying assignment or reaches the conclusion NO. Here is a brief and somewhat informal description.

Initialization. Set $A_1 = A_2 = \ldots = A_m = \emptyset$ and let $I_1, \ldots, I_m$ be arbitrary q-tuples of variable indices.
Loop. (Repeat the following steps until termination.)
- Set C_j to be the union of the relations from $\mathcal{R}$ violated by every tuple in A_j ($j = 1, \ldots, m$).
- Call algorithm $\mathcal{A}$ for $\{C_1, \ldots, C_m\}$. Return NO if $\mathcal{A}$ returned NO. Otherwise let $a = (a_1, \ldots, a_\ell)$ be the assignment returned by $\mathcal{A}$.
- Call oracle V with assignment a. Return a if V accepted it. Otherwise let j and $I = (j_1, \ldots, j_q)$ be the constraint index resp. the variable index array revealed by V.
- Set $I_j = I$, add $a' = (a_{j_1}, \ldots, a_{j_q})$ to A_j and continue loop.

In the following more detailed description, we indicate the actual repetition number in upper indices. The instance $\mathcal{C}^t = \{C_1^t, \ldots, C_m^t\}$ of the tth call of $\mathcal{A}$ is defined as $C_j^t = \bigcup_{R \in \mathcal{R}: R \cap A_j^t = \emptyset} R(x_{j_1^t}, \ldots, x_{j_q^t})$ where $A_j^t \subseteq W_q$ and $I_j = (j_1^t, \ldots, j_q^t) \in [\ell]^{(q)}$, for $j \in [m]$, are determined successively by $\mathcal{B}$. The set A_j^t reflects the algorithm's knowledge after t steps: it contains those q-tuples which, at that instant, are known to be violating the jth constraint. Initially $A_j^1 = \emptyset$ and $(j_1^1, \ldots, j_q^1)$ is arbitrary. If the output of $\mathcal{A}$ for $\mathcal{C}^t$ is NO then $\mathcal{B}$ outputs NO. If the output of $\mathcal{A}$ for $\mathcal{C}^t$ is $a \in W$ then $\mathcal{B}$ submits a to the $\{\mathcal{V}\}$-revealing oracle V. If V answers YES then $\mathcal{B}$ outputs a. If the oracle does not find a satisfying, and reveals j and $(j_1, \ldots, j_q)$ about

[1] That the assignment length ℓ is a function of n is, as far as we can see, quite standard in the literature.

the violated constraint, then $\mathcal{B}$ does not change A_i^t and $(i_1^t, \ldots, i_q^t)$ for $i \neq j$, but sets $A_j^{t+1} = A_j^t \bigcup \{(a_{j_1}, \ldots, a_{j_q})\}$, and $(j_1^{t+1}, \ldots, j_q^{t+1}) = (j_1, \ldots, j_q)$. Observe that the q-tuple for the jth constraint is changed at most once, the first time when the revealing oracle gives the index of the jth constraint.

To prove that the algorithm correctly solves H–$S_{\{\mathcal{V}\}}$, let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be an instance of S and let V be any $\{\mathcal{V}\}$-revealing oracle for $\mathcal{C}$. We have to show that if $\mathcal{B}$ answers NO then $\mathcal{C}$ is unsatisfiable. If $\mathcal{B}$ answers NO, then for some t, the tth call of $\mathcal{A}$ resulted in output NO. By construction, A_j^t and $(j_1^t, \ldots, j_q^t)$ are such that for every $j \in [m]$, if $R \cap A_j^t \neq \emptyset$ then the relation of C_j can't be $R(x_{j_1^t}, \ldots, x_{j_q^t})$. Indeed, if $C_j = R(x_{j_1^t}, \ldots, x_{j_q^t})$ and $b \in R \cap A_j^t$ then at the call when b was added to A_j^t the oracle's answer is incorrect. Therefore all possible remaining relations for C_js are included in C_j^t, and since $\mathcal{C}^t$ is unsatisfiable, so is $\mathcal{C}$.

For the complexity of the algorithm let us remark that if for some j and t, the constraint C_j^t is the empty relation then $\mathcal{B}$ stops since $\mathcal{C}^t$ becomes unsatisfiable. This happens in particular if $A_j^t = W_q$. Since for every call to $\mathcal{A}$ one new element is added to one of the A_j^t and at least one new relation in $\mathcal{R}$ is excluded from C_j^t, the number of calls is upper bounded by $m \times \min\{\dim(\mathcal{R}), |W_q|\}$. To compute a new constraint, some number of relations in $\mathcal{R}$ have to be computed on a new argument, which can be done in time $s \times \mathrm{comp}(\mathcal{R})$.

We now prove (b). Let $\mathcal{A}$ be an algorithm which solves H–$S_{\{\mathcal{V}\}}$ in time T. Without loss of generality we suppose that $\mathcal{A}$ only outputs a satisfying assignment a after submitting it to the verifying oracle. We define an algorithm $\mathcal{B}$ for $\bigcup$S. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be an instance of $\bigcup$S where for $j \in [m]$, $C_j = \bigcup_{R \in \mathcal{R}_j} R(x_{j_1}, \ldots, x_{j_q})$, for some $\mathcal{R}_j \subseteq \mathcal{R}$ and $I_j = (j_1, \ldots, j_q) \in [\ell]^{(q)}$. The algorithm $\mathcal{B}$ runs $\mathcal{A}$, and outputs NO whenever $\mathcal{A}$ outputs NO. During $\mathcal{A}$'s run $\mathcal{B}$ simulates a $\{\mathcal{V}\}$-revealing oracle V for $\mathcal{A}$. Here is rather informal description of $\mathcal{B}$.

Initialization. Set $A_1 = A_2 = \ldots = A_m = \emptyset$ and run $\mathcal{A}$ until it calls oracle V first time.
Loop. (Repeat the following steps until termination.)
- Let $a = (a_1, \ldots, a_\ell)$ be the assignment with which $\mathcal{A}$ calls V. Check if a satisfies $\mathcal{C}$. Return a if yes.
- Choose index j such that a violates C_j and add tuple $(a_{j_1}, \ldots, a_{j_q})$ to A_j.
- Run subsequent steps of $\mathcal{A}$ (with j and $(j_1, \ldots, j_q)$ as revealed information) until the next oracle call.

Now we give more details about how $\mathcal{B}$ implements V. Simultaneously with V's description, for $t \geq 1$, we also specify instances $\mathcal{C}^t = \{C_1^t, \ldots, C_m^t\}$ of $\bigcup$S which will be used in the proof of correctness of the algorithm. For $j \in [m]$, the constraints of $\mathcal{C}^t$ are defined as $C_j^t = \bigcup_{R \in \mathcal{R}_j : R \cap A_j^t = \emptyset} R(x_{j_1}, \ldots, x_{j_q})$, where the sets $A_j^t \subseteq W_q$ are determined by the result of the tth call to the oracle. Initially $A_j^0 = \emptyset$. For the tth request $a \in W$, the algorithm $\mathcal{B}$ checks if a satisfies $\mathcal{C}$. If it is the case then V returns YES and $\mathcal{B}$ outputs a. Otherwise there exists $j \in [m]$ such that a violates C_j, and the answer of the oracle is j and $(j_1, \ldots, j_q)$ (where j can be chosen arbitrarily among the violated constraints, if there are several). Observe that this is a legitimate oracle for any instance of H–$S_{\{\mathcal{V}\}}$ whose jth constraint is arbitrarily chosen from $\mathcal{R}_j$. We define $A_j^t = A_j^{t-1} \bigcup \{(a_{j_1}, \ldots, a_{j_q})\}$, and for $i \neq j$ we set $A_i^t = A_i^{t-1}$.

To show the correctness of $\mathcal{B}$, we prove that whenever $\mathcal{A}$ outputs NO, the instance $\mathcal{C}$ is unsatisfiable. Let us suppose that $\mathcal{A}$ made t queries before outputting NO. An algorithm for H–$S_{\{\mathcal{V}\}}$ can output NO only if all possible instances of S which are compatible with the answers received from the oracle are unsatisfiable. In such an instance the relation of the jth constraint has necessarily empty intersection with A_j^t, therefore we can deduce that the $\bigcup$S instance $\mathcal{C}^t$ is unsatisfiable. It also holds that $A_j^t \bigcap (\bigcup_{R \in \mathcal{R}_j} R) = \emptyset$ for every $j \in [m]$, since if $b \in A_j^t \bigcap (\bigcup_{R \in \mathcal{R}_j} R)$ then the request to the oracle that caused b to be added to A_j^t wouldn't violate the jth constraint. Thus $\bigcup_{R \in \mathcal{R}_j} R \subseteq \bigcup_{R \in \mathcal{R} : R \cap A_j^t = \emptyset} R$, and $\mathcal{C}$ is unsatisfiable.

For the complexity analysis we observe that during the algorithm, for every query to the oracle and for every constraint, one relation in $\bigcup \mathcal{R}$ is evaluated. □

Theorem 3.2. *(a) If* E–S *is solvable in time T then* H–$S_{\{\mathcal{R}\}}$ *is solvable in time $O((T + |[\ell]^{(q)}| \times \mathrm{comp}(\mathcal{R})) \times m \times |[\ell]^{(q)}|)$.*
(b) If H–$S_{\{\mathcal{R}\}}$ *is solvable in time T then* E–S *is solvable in time $O(T \times m \times \mathrm{comp}(\text{E–}\mathcal{R}))$.*

Like above, instances of H–$S_{\{\mathcal{R}\}}$ consisting of m relations correspond to instances of E–S also consisting of m relations.

Proof. The proof is similar to the proof of Theorem 3.1. We first prove (a). Let $\mathcal{A}$ be an algorithm which solves E–S in time T. We define an algorithm $\mathcal{B}$ for H–$S_{\{\mathcal{R}\}}$. The algorithm will repeatedly call $\mathcal{A}$, until it finds a satisfying assignment or reaches the conclusion NO. Since each constraint of E–S is an ℓ-ary relation, we can identify it with the relation itself. Here is again a brief and informal description of $\mathcal{B}$.

Initialization. Set $A_1 = A_2 = \ldots = A_m = \emptyset$ and $C_1 = C_2 = \ldots = C_m = W$.
Loop. (Repeat the following steps until termination.)
- Call $\mathcal{A}$ for $(C_1, \ldots, C_m)$. Return NO if $\mathcal{A}$ returned NO. Otherwise let $a = (a_1, \ldots, a_\ell)$ be the assignment output by $\mathcal{A}$.

- Call oracle V with assignment a. Return a if V accepted it. Otherwise let j and R be the constraint index resp. the relation revealed by V.
- Add $a = (a_{j_1}, \ldots, a_{j_\ell})$ to A_j, let I be the set of the variable index arrays $(j_1, \ldots, j_q)$ such that $R^{(j_1,\ldots,j_q)}$ is violated by every tuple in A_j, update C_j to be R^I, the union of the relations $R^{(j_1,\ldots,j_q)}$ over $(j_1, \ldots, j_q) \in I$ and continue loop.

In the more detailed description and analysis we again apply repetition indices. For the first call $\mathcal{C}^1 = \{C_1^1, \ldots, C_m^1\}$ we set $C_j^1 = W$. For $t > 1$, the instance of the tth call will be defined recursively via $A_j^t \subseteq W$ and $I_j^t \subseteq [\ell]^{(q)}$, for $j \in [m]$, where initially we set $A_1^1 = \ldots = A_m^1 = \emptyset$ and $I_1^1 = \ldots = I_m^1 = [\ell]^{(q)}$. Here the set I_j^t reflects the algorithm's knowledge after t steps: it contains those q-tuples of indices which, at that instant, can still be the variable indices of the jth constraint. If the output of $\mathcal{A}$ for $\mathcal{C}^{t-1}$ is NO then $\mathcal{B}$ outputs NO. If the output of $\mathcal{A}$ for $\mathcal{C}^{t-1}$ is $a \in W$ then $\mathcal{B}$ submits a to the $\{\mathcal{R}\}$-revealing oracle V. If V answers YES then $\mathcal{B}$ outputs a. If the oracle does not find a satisfying, and reveals j and $R \in \mathcal{R}$ about the violated constraint, then $\mathcal{B}$ does not change A_i^{t-1}, I_i^{t-1} and C_i^{t-1} for $i \neq j$, but sets $A_j^t = A_j^{t-1} \bigcup \{a\}$ and $I_j^t = \{(j_1, \ldots, j_q) : A_j^t \bigcap R^{(j_1,\ldots,j_q)} = \emptyset\}$. Finally we define $C_j^t = R^{I_j^t}$.

To prove that the algorithm correctly solves H–S$_{\{\mathcal{R}\}}$, let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be an instance of S and let V be any $\{\mathcal{R}\}$-revealing oracle for $\mathcal{C}$. We have to show that if $\mathcal{B}$ answers NO then $\mathcal{C}$ is unsatisfiable. If $\mathcal{B}$ answers NO then for some t, the tth call of $\mathcal{A}$ resulted in output NO. We claim that for every constraint C_j whose relation R has been already revealed, if $R^{(j_1,\ldots,j_q)} \cap A_j^t \neq \emptyset$ then C_j can not be $R(x_{j_1}, \ldots, x_{j_q})$. Indeed, if $C_j = R^{(j_1,\ldots,j_q)}(x_{j_1}, \ldots, x_{j_q})$ and $a \in R \cap A_j^t$ then at the call when a was added to A_j^t the oracle answer is incorrect. Therefore C_j^t is the union, over all still possible variable index q-tuples $(j_1, \ldots, j_q)$, of $R^{(j_1,\ldots,j_q)}$. Since $\mathcal{C}^t$ is unsatisfiable, so is $\mathcal{C}$.

For the complexity of the algorithm let us remark that if for some j and t, the constraint C_j^t is the empty relation then $\mathcal{B}$ stops since $\mathcal{C}^t$ becomes unsatisfiable. This happens in particular if $I_j^t = \emptyset$. Since for every call to $\mathcal{A}$, for some j, the size of I_j^t decreases by at least one, the total number of calls is upper bounded by $m \times |[\ell]^{(q)}|$. To compute a new constraints, at most $|[\ell]^{(q)}|$ relations from $\mathcal{R}$ evaluated in a new argument. Therefore the overall complexity is as claimed.

We now prove (b). Let $\mathcal{A}$ be an algorithm which solves H–S$_{\{\mathcal{R}\}}$ in time T. Without loss of generality we suppose that $\mathcal{A}$ only outputs a satisfying assignment a after submitting it to the verifying oracle. We define an algorithm $\mathcal{B}$ for E–S. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be an instance of E–S where for $j \in [m]$, we have $C_j = R_{k_j}^{I_j}$ for some $R_{k_j} \in \mathcal{R}$ and $I_j \subseteq [\ell]^{(q)}$. The algorithm $\mathcal{B}$ runs $\mathcal{A}$, and outputs NO whenever $\mathcal{A}$ outputs NO. During $\mathcal{A}$'s run $\mathcal{B}$ simulates an $\{\mathcal{R}\}$-revealing oracle V for $\mathcal{A}$. Here is an informal description of $\mathcal{B}$.

Initialization. Set $A_1 = A_2 = \ldots = A_m = \emptyset$. Run $\mathcal{A}$ until it calls V first time.
Loop. (Repeat the following steps until termination.)
- Let a be the assignment that $\mathcal{A}$ submits to V. Return a if it satisfies $\mathcal{C}$.
- Choose and index j such that a violates C_j.
- Run subsequent steps of $\mathcal{A}$ with revealed information j and R_{k_j} until the next oracle call.

Now we give more details. Simultaneously with V's description, for $t \geq 1$, we also specify instances $\mathcal{C}^t = \{C_1^t, \ldots, C_m^t\}$ of E–S which will be used in the proof of correctness of the algorithm. Again we identify the ℓ-ary constraints with their relations. The constraints of $\mathcal{C}^t$ are set to be $C_j^t = R_{k_j}^{I_j^t}$, where the sets $I_j^t \subseteq [\ell]^{(q)}$ are defined as $I_j^t = \{(j_1, \ldots, j_q) : A_j^t \bigcap R_{k_j}^{(j_1,\ldots,j_q)} = \emptyset\}$, and the sets $A_j^t \subseteq W$ are determined by the result of the tth call to the oracle. Initially $A_j^0 = \emptyset$. For the tth request $a \in W$, the algorithm $\mathcal{B}$ checks if a satisfies $\mathcal{C}$. If it is the case then V returns a and $\mathcal{B}$ outputs a. Otherwise there exists $j \in [m]$ such that a violates C_j, and the answer of the oracle is j and R_{k_j}. Observe that this is a legitimate oracle for any instance of H–S$_{\{\mathcal{R}\}}$ whose jth constraint is arbitrarily chosen from $\{R_{k_j}(x_{j_1}, \ldots, x_{j_q}) : (j_1, \ldots, j_q) \in I_j\}$. We define $A_j^t = A_j^{t-1} \bigcup \{a\}$, and for $i \neq j$ we set $A_i^t = A_i^{t-1}$.

To show the correctness of $\mathcal{B}$, we prove that whenever $\mathcal{A}$ outputs NO, the instance $\mathcal{C}$ is unsatisfiable. Let us suppose that $\mathcal{A}$ made t queries before outputting NO. An algorithm for H–S$_{\{\mathcal{R}\}}$ can output NO only of all possible instances of S which are compatible with the answers received from the oracle are unsatisfiable. In such an instance the jth constraint has necessarily empty intersection with A_j^t, therefore we can deduce that the E–S instance $\mathcal{C}^t$ is unsatisfiable. It also holds that $A_j^t \bigcap C_j = \emptyset$ for every $j \in [m]$, since if $a \in A_j^t \bigcap C_j$ then the request to the oracle that caused a to be added to A_j^t wouldn't violate the jth constraint. Thus $C_j \subseteq C_j^t$, and $\mathcal{C}$ is unsatisfiable.

For the complexity analysis we just have to observe that during the algorithm, for every query to the oracle and for every constraint, one relation in E–$\mathcal{R}$ is evaluated. □

Theorem 3.3. *(a) If* $\bigcup$X–S *is solvable in time* T *then* H–S$_\emptyset$ *is solvable in time* $O((T + s \times \frac{\ell!}{(\ell-q)!} \times \mathrm{comp}(\mathcal{R})) \times m \times \dim(\bigcup \text{X–}\mathcal{R}))$.
(b) If H–S$_\emptyset$ *is solvable in time* T *then* $\bigcup$X–S *is solvable in time* $O(T \times m \times \mathrm{comp}(\bigcup \text{X–}\mathcal{R}))$.

Again, instances of H–S$_\emptyset$ consisting of m relations correspond to instances of $\bigcup$X–(S) also consisting of m relations.

Proof. Apply Theorem 3.1 to X–S and observe that H–X–$S_{\{\mathcal{V}\}}$ and H–$S_{\emptyset}$ are essentially the same in the sense that an algorithm solving one of the problems also solves the other one. Indeed, the variable index disclosure of the $\{\mathcal{V}\}$-revealing oracle is pointless since the relations in X–S involve all variables. Moreover, the map sending a constraint $R(x_{j_1}, \dots x_{j_q})$ of S to the constraint $R^{(j_1,\dots,j_q)}(x_1, \dots x_\ell)$ of X–S is a bijection which preserves satisfying assignments. □

Corollary 3.4. *Let* comp($\mathcal{R}$) *be polynomial. Then the complexities of the following problems are polynomial time equivalent: (a)* H–$S_{\{\mathcal{V}\}}$ *and* $\bigcup$S *if the number of relations s is constant, (b)* H–$S_{\{\mathcal{R}\}}$ *and* E–S *if the arity q is constant, (c)* H–$S_{\emptyset}$ *and* $\bigcup$X–S *if both s and q are constant.*

The polynomial time equivalences of Theorems 3.1, 3.2, 3.3 and Corollary 3.4 remain true when the algorithms have access to the same computational oracle. Therefore, we get generic easiness results for H–CSPs under an NP oracle.

We also remark that the number of iterations in algorithms for the transfer theorems give generic upper bounds on the *trial complexity* of the hidden CSPs In the constraint index and variable revealing model this bound is $m \times \min\{\dim \bigcup \mathcal{R}, |W_q|\}$, in the constraint index and relation revealing model $m \times [\ell]^{(q)}$ oracle calls are sufficient, while in the constraint index revealing model we obtain the bound $m \times \dim \bigcup X - \mathcal{R}$.

4. Constraint-index and variables revealing oracle

In this section, we present some applications of our transfer theorem when the index of the constraint and the variables participating in that constraint are revealed. We consider the following CSPs. In the descriptions below, unless explicitly specified, W is the full domain $[w]^\ell$.

1. Deltas on Triplets (Δ): Formally, $w = 2$, $q = 3$, and $\mathcal{R} = \{R_{abc} : \{0,1\}^3 \to \{\mathrm{T}, \mathrm{F}\} \mid a, b, c \in \{0,1\}\}$, where $R_{abc}(x, y, z) := (x = a) \wedge (y = b) \wedge (z = c)$.
2. Hyperplane Non-Cover (HYP–NC): Let p be a prime, and F_p be the field of size p. Denote $V = F_p^N$, and $S = \{\text{all hyperplanes in } F_p^N\}$. Informally, given a set of hyperplanes $S' \subseteq S$, the problem asks to decide if there exists $v \in F_p^N$ not covered by these hyperplanes. Formally, $\ell = 1$, $q = 1$, $w = p^N$, $W = V$ and $\mathcal{R}_S = \{R_H \mid H \in S\}$ where $R_H(a)$ evaluates to T if and only if $a \notin H$.
3. Arbitrary sets of binary relations on Boolean alphabet, (in particular, 2SAT): Formally for 2SAT, we have $w = 2$, $q = 2$, and $\mathcal{R} = \{R_T, R_F, R_a, R_b, R_{\neg a}, R_{\neg b}, R_{a\vee b}, R_{a\vee\neg b}, R_{\neg a\vee b}, R_{\neg a\vee\neg b}\}$, where for $(\alpha, \beta) \in \{\mathrm{T}, \mathrm{F}\}^q$, $R_T(\alpha, \beta) := \mathrm{T}$, $R_F(\alpha, \beta) := \mathrm{F}$, $R_a(\alpha, \beta) := \alpha$, $R_b(\alpha, \beta) := \beta$, $R_{\neg a}(\alpha, \beta) := \neg\alpha$, $R_{\neg b}(\alpha, \beta) := \neg\beta$, $R_{a\vee b}(\alpha, \beta) := \alpha \vee \beta$, $R_{a\vee\neg b}(\alpha, \beta) := \alpha \vee \neg\beta$, $R_{\neg a\vee b}(\alpha, \beta) := \neg\alpha \vee \beta$, $R_{\neg a\vee\neg b}(\alpha, \beta) := \neg\alpha \vee \beta$.
4. Exact-Unique Game Problem (UG[k]): Given an undirected graph, $G = (V, E)$, and a permutation $\pi_e : [\![k]\!] \to [\![k]\!]$ for every edge $e \in E$, the goal is to decide if one can assign labels $\alpha_v \in [\![k]\!]$ for every vertex $v \in V$ *s.t.* for every edge $e = \{u, v\} \in E$ with $u < v$ we have $\pi_e(\alpha_u) = \alpha_v$. Formally: $w = k$, $q = 2$ and $\mathcal{R} = \{\pi : [\![k]\!] \to [\![k]\!] \mid \pi \text{ is a permutation}\}$.
5. k-Clique Isomorphism (kCLQ $-\,-$ ISO):
   Given an undirected graph $G = (V, E)$, determine if there exists a permutation π on $[n]$ *s.t.*
   (1) $\forall (i, j) \in E$, $R_{\leq k}(\pi(i), \pi(j))$;
   (2) $\forall (i, j) \notin E$, $\neg R_{\leq k}(\pi(i), \pi(j))$.
   Formally, $w = n$, $q = 2$, $\ell = n$, W is the set of n-tuples of integers from $[n]$ which define permutations on $[n]$, and $\mathcal{R} = \{R_{\leq k}, \neg R_{\leq k}\}$, where $R_{\leq k}(\alpha, \beta) := \mathrm{T} \iff \alpha \leq k \,\&\, \beta \leq k$.
6. Equality to some member in a fixed class of graphs ($\mathsf{EQ}_{\mathcal{K}}$): For a fixed class $\mathcal{K}$ of graphs on n vertices variables, we denote by $\mathcal{P}_{\mathcal{K}} : \{0,1\}^{\binom{n}{2}} \to \{T, F\}$ the property of being equal to a graph from $\mathcal{K}$. We assume that graphs are represented by tuples from $\{0,1\}^{\binom{n}{2}}$. Formally, $W = \mathcal{K}$, $w = 2$, $q = 1$, $\ell = \binom{n}{2}$, and $\mathcal{R} = \{\mathsf{Id}, \mathsf{Neg}\}$. Here we assume that membership in $\mathcal{K}$ can be tested in polynomial time. We will consider the following special cases:
   - Equality to k-Clique ($\mathsf{EQ}_{k\mathsf{CLQ}}$): Given a graph, decide if it is equal to a k-clique.
   - Equality to Hamiltonian Cycle ($\mathsf{EQ}_{\mathsf{HAMC}}$): Decide if G is a cycle on all n vertices.
   - Equality to Spanning Tree ($\mathsf{EQ}_{\mathsf{ST}}$): Given a graph, decide if it is a spanning tree.

We have seen in the Introduction that the hidden version of 1SAT in the constraint index revealing model is NP-hard. Here we will show that if the variables are also revealed, even the hidden version of 2SAT becomes solvable in polynomial time. Deltas on triplets will provide a simple example of ternary Boolean constraints for which the "normal" satisfaction problem can be solved in polynomial time but the hidden version becomes NP-hard. Hyperplane Non-Cover over the two-element field is equivalent to a system of linear equations. Interestingly, its hidden version will turn out to be NP-hard. The unique game problem is a prominent CSP problem, whose approximate version has been studied intensively since [8]. It is known that the exact version is in P for any k, and we show that it is only easy in the trial and error model for $k = 2$. Isomorphisms with cliques is a problem considered in [2]. We included it to demonstrate how easy to prove hardness of its hidden version based on the transfer theorem. Also note that our hardness result is somewhat stronger than that of [2] as the latter is proved for the constraint index revealing model while here more information is revealed. A formally different, although logically equivalent formulation of the same problem is equality with a k-clique. The hardness result can be

extended to equalities with other distinguished graphs, like Hamiltonian circles. However, equality with spanning trees will remain easy.

Theorem 4.1. *The following problems can be solved in polynomial time: (a)* H–2SAT$_{\{\mathcal{V}\}}$, *(b)* H–UG[2]$_{\{\mathcal{V}\}}$, *(c)* H–EQ$_{\mathsf{ST}\{\mathcal{V}\}}$.

Theorem 4.2. *The following are* NP*-hard: (a)* H–$\Delta_{\{\mathcal{V}\}}$, *(b)* H–HYP–NC$_{\{\mathcal{V}\}}$, *(c)* H–UG[k]$_{\{\mathcal{V}\}}$ *for* $k \geq 3$, *(d)* H–kCLQ $-$ $-$ISO$_{\{\mathcal{V}\}}$ *for* $0.1n \leq k \leq 0.9n$, *(e)* H–EQ$_{k\mathsf{CLQ}\{\mathcal{V}\}}$ *for* $0.1n \leq k \leq 0.9n$, *(f)* H–EQ$_{\mathsf{HAMC}\{\mathcal{V}\}}$.

Proof of Theorem 4.1. We show that the following problems in the hidden model with the constraint and variable index revealing oracle are solvable in polynomial time.

(a) Arbitrary binary Boolean relations (H–2SAT$_{\{\mathcal{V}\}}$) In the case of 2SAT, taking the union of any two relations in $\mathcal{R}_{2\mathsf{SAT}}$ is equivalent to the disjunction of the two boolean expressions the relations signify. For example, $R_a \bigcup R_b = R_{a\vee b}$ and the union remains in $\mathcal{R}_{2\mathsf{SAT}}$. Hence, $\bigcup 2\mathsf{SAT} = 2\mathsf{SAT}$, which is in P. Therefore, from Theorem 3.1(a), H–2SAT$_{\{\mathcal{V}\}}$ is also in P. The above statement can be extended to an *arbitrary* set $\mathcal{R}'$ of binary relations as follows. Let $\mathcal{R}''$ stand for the set of *all* binary relations in Boolean variables. We trivially have $\bigcup \mathcal{R}' \subseteq \mathcal{R}''$, therefore an instance of H–2SAT$_{\{\mathcal{V}\}}$ can actually be described by a conjunction of the form $\bigwedge_{k=1}^{m} R_k(x_{i_k}, x_{j_k})$ where R_k is a binary relation. Expressing each R_k by a Boolean formula in conjunctive normal form, we obtain an instance of 2SAT consisting of $O(m)$ clauses, which can be solved in polynomial time.

(b) Unique Games (H–UG[2]$_{\{\mathcal{V}\}}$) UG[2] is a CSP with $w = 2$, $q = 2$, and $\mathcal{R} = \{\pi : [\![2]\!] \to [\![2]\!] \mid \pi \text{ is a permutation}\}$. The only permutations in $\mathcal{R}_{\mathsf{UG[2]}}$ enforce that either $\alpha_u = \alpha_v$ or $\alpha_u = \alpha_v \oplus 1$ for an edge $e = (u, v)$. Both of these relations can be represented as binary boolean relations. Hence, UG[2] is an instance of 2SAT and from Theorem 4.1(*a*), H–UG[2]$_{\{\mathcal{V}\}}$ is in P.

(c) Equality/Isomorphism to a member in a fixed class of graphs We define the H–EQ$_{\mathcal{K}\{\mathcal{V}\}}$ problem in more detail. Let $\mathcal{K}$ be a class of graphs on n vertices. We define $\mathcal{P}_{\mathcal{K}} : \{0,1\}^{\binom{n}{2}} \to \{T, F\}$ as the graph property of being equal to a graph from $\mathcal{K}$. Correspondingly, $W = \mathcal{K}$.

Formally, for $\mathcal{P}_{\mathcal{K}}$ we consider the CSP EQ$_{\mathcal{K}}$ with $w = 2$, $q = 1$, $W = \{\alpha \in \{0,1\}^{\binom{n}{2}} \mid \alpha \in \mathcal{K}\}$, $\ell = \binom{n}{2}$, and $\mathcal{R} = \{\mathsf{Id}, \neg\}$. Given a graph instance $G = (V, E)$ in this model, the $\binom{n}{2}$ constraints for G_2 are such that $C_e = \mathsf{Id}(\alpha_e)$ for $e \in E$ and $C_e = \neg(\alpha_e)$ otherwise. This implies that $\bigcup \mathcal{R} = \{\mathsf{Id}, \neg, \mathsf{T}\}$ and instances of $\bigcup \mathsf{EQ}_{\mathcal{K}}$ are parametrized with graphs (sets of edges) $E_1 \subseteq E_2$. Here E_2 is the set of unordered pairs e for which the constraint is either Id or T while E_1 consists of pairs for which the constraint is Id. The $\bigcup \mathsf{EQ}_{\mathcal{K}}$-problem becomes then: given sets E_1, E_2 such that $E_1 \subseteq E_2$, does there exist a graph $G' = (V, E') \in \mathcal{K}$ such that $E_1 \subseteq E' \subseteq E_2$?

From Theorem 3.1, the complexity of H–EQ$_{\mathcal{K}}$, can be analyzed by considering the complexity of $\bigcup \mathsf{EQ}_{\mathcal{K}}$. Below, we analyze the complexity of $\bigcup \mathsf{EQ}_{\mathcal{K}}$ when $\mathcal{K}$ is the class of spanning trees on n vertices.

Remark 4.3. For any $\mathcal{K}$, if we take $E_1 = \emptyset$, then solving EQ$_{\mathcal{K}}$ becomes equivalent to finding out if there exists $G \in \mathcal{K}$ which is a subgraph of E_2.

Remark 4.4. Note that if we assume that $\mathcal{K}$ is the set of all graphs isomorphic to some G_0 and $E_1 = E_2$ as arbitrary graphs on n vertices, then solving EQ$_{\mathcal{K}}$ becomes equivalent to finding out if E_2 is isomorphic to G_0.

Proof for Equality to a Spanning Tree (H–EQ$_{\mathsf{ST}\{\mathcal{V}\}}$): Here, $\mathcal{K}$ is the set of all possible spanning trees on n vertices and E_1 without loss of generality is a forest F. E_2 is any arbitrary graph on n vertices containing E_1. In this case, the $\bigcup \mathsf{EQ}_{\mathcal{K}}$ problem becomes equivalent to finding a spanning tree on E_2 which also contains the forest F. This problem is in P which implies that the H–EQ$_{\mathsf{ST}\{\mathcal{V}\}}$ problem is also in P.

This completes the proof for Theorem 4.1. □

Proof of Theorem 4.2. We show that the following problems in the hidden model with the constraint and variable index revealing oracle are NP-hard. Using Theorem 3.1, the complexity of each H–S$_{\{\mathcal{V}\}}$ is analyzed by considering the complexity of $\bigcup$S.

(a) Deltas on Triplets (H–$\Delta_{\{\mathcal{V}\}}$) By definition, each relation in $\mathcal{R}_{\Delta}$ identifies a boolean string on 3 variables. This implies that $\bigcup \mathcal{R}_{\Delta}$ forms the set of all Boolean predicates on 3 variables. Thus, 3SAT can be expressed as the $\bigcup \Delta$ problem. Hence, from Theorem 3.1(b), H–$\Delta_{\{\mathcal{V}\}}$ is NP-hard.

(b) Hyperplane Non-Cover (H–HYP–NC$_{\{\mathcal{V}\}}$) The Hyperplane Non-Cover problem (HYP–NC) is the solvability of homogeneous linear in-equations in F_p^N. The HYP $-$ NC problem over Z_p^N for $p \geq 3$ includes the 3COL problem and is already NP-hard. To see this, let E be a graph on vertex set $[N]$ and consider the in-equations $x_i \neq x_j$ for indices $i, j \in [N]$ such that $\{i, j\} \in E$. If $p = 3$ that's all we need. Otherwise, add a variable y together with the in-equations $y \neq 0$, $x_i \neq ky$ for $i \in [N]$ and $k \in [p-3]$.

Hence, it remains to consider the H–HYP–NC$_{\{\mathcal{V}\}}$ problem over F_2^N, which we need to examine $\bigcup$HYP–NC by Theorem 3.1. In this setting, let T be the set of all subspaces (not necessarily hyperplanes) of F_2^N. Then the set of constraints of $\bigcup$HYP–NC consists of $\{R_P \mid P \in T\}$ where $R_P(a)$ evaluates to T if and only if $a \notin P$. This problem is NP-hard, as it includes non-covering by subspaces of codimension 2 which encompasses the 4COL problem. Hence, the former will be NP-hard using Theorem 3.1.

(c) Unique games (H–UG[k]$_{\{\mathcal{V}\}}$ for $k \geq 3$) UG[3] is a CSP with $w = 3$, $q = 2$, and $\mathcal{R} = \{\pi : [\![3]\!] \to [\![3]\!] \mid \pi \text{ is a permutation}\}$. Let

$$R^\circ := \bigcup_{\pi:(\forall i)(\pi(i)\neq i)} \pi .$$

Note that $R^\circ \in \bigcup \mathcal{R}$. Choosing R° as the constraint for every edge gives us the 3COL problem. Hence, from Theorem 3.1(b), H–UG[3]$_{\{\mathcal{V}\}}$ is NP-hard.

Remark 4.5. Our proof method also shows that H–UG[k]$_{\{\mathcal{V}\}}$ is NP-hard for any $k > 2$.

(d) k-Clique Isomorphism (H–kCLQ $-\,-$ISO$_{\{\mathcal{V}\}}$) for $0.1n \leq k \leq 0.9n$ Obviously, replacing the constraints of type $R_{\leq k}$ by $R_{\leq k} \cup \neg R_{\leq k}$ in an instance of H–kCLQ $-\,-$ISO we obtain an instance of $\bigcup$H–kCLQ $-\,-$ISO. This however just means omitting Constraints (1), and we obtain the kCLQ problem (deciding whether the graph contains a k-clique) which is NP-hard. Hence from Theorem 3.1(b), the H–kCLQ $-\,-$ISO$_{\{\mathcal{V}\}}$ problem is NP-hard.

(e) Equality to a k-Clique (H–EQ$_{k\text{CLQ}\{\mathcal{V}\}}$) for $0.1n \leq k \leq 0.9n$ We use the framework defined in the previous proof for the H–EQ$_{\text{ST}\{\mathcal{V}\}}$ problem. As mentioned in Remark 4.3, given a graph E_2, consider $\mathcal{K}$ to be the set of all possible k-cliques on n vertices and $E_1 = \emptyset$. In this setting, the $\bigcup$EQ$_{\mathcal{K}}$ problem is equivalent to finding a k-clique on E_2 which is NP-hard.

Remark 4.6. The above proof could also serve as an alternate proof for Theorem 4.2(d).

(f) Equality to a Hamiltonian Cycle (H–EQ$_{\text{HAMC}\{\mathcal{V}\}}$) We use the framework defined in the previous proof for the H–EQ$_{\text{ST}\{\mathcal{V}\}}$ problem. Here, $\mathcal{K}$ is the set of all possible Hamiltonian cycles on n vertices and $E_1 = \emptyset$. For an arbitrary graph E_2, the $\bigcup$EQ$_{\mathcal{K}}$ problem parametrized by E_1 and E_2 becomes equivalent to deciding if E_2 has a Hamiltonian cycle, which is NP-hard.

This completes the proof for Theorem 4.2. □

5. Constraint-index and relation revealing oracle

Theorem 5.1. *Let* S *be a* CSP *with constant arity q and constant alphabet size w. Assume that for every $\alpha \in [\![w]\!]$, there is a non-empty relation $R_\alpha \in \mathcal{R}$ such that $(\alpha, \ldots, \alpha) \notin R_\alpha$. Then,* H–S$_{\{\mathcal{R}\}}$ *is* NP*-hard.*

Proof. We show that E–S is NP-hard. We will reduce to it the problem E–3SAT which consists of those instances of 3SAT where in each clause either every variable is positive, or every variable is negated. Restricting to these instances of 3SAT is known as MONSAT, whose NP-completeness can be deduced, for example, from Schaefer's characterization [10].

We first extend the relations for S as follows. Let $q' = (w-1)q + 1$ and let $\mathcal{R}' \subseteq [\![w]\!]^{q'}$ be the set of q'-ary relations that can be obtained as an extension of an element of $\mathcal{R} \setminus \{\emptyset\}$ from any q coordinates. Since q and w are constant, the cardinality of $\mathcal{R}'$ is also constant. We claim that $\bigcap_{R\in\mathcal{R}'} R = \emptyset$. Indeed, every $a \in [\![w]\!]^{q'}$ has a sub-sequence $(\alpha, \ldots, \alpha)$ of length q for some $\alpha \in [\![w]\!]$, therefore the extension of R_α from these q coordinates does not contain a. Let $\{R^0, R^1, \ldots, R^h\}$ be a minimal subset of $\mathcal{R}'$ such that $\bigcap_{i=0}^h R^i = \emptyset$. Since the empty relation is not in $\mathcal{R}'$, we have $h \geq 1$. Let us set $A^0 = \bigcap_{i\neq 1} R^i$ and $A^1 = \bigcap_{i\neq 0} R^i$. Then $A^0 \bigcap A^1 = \emptyset$, and because of the minimality condition, $A^0 \neq \emptyset$ and $A^1 \neq \emptyset$.

For a boolean variable x, we will use the notation $x^1 = x$ and $x^0 = \bar{x}$. The main idea of the proof is to encode a boolean variable x^1 by the relation A^1 and x^0 by A^0. We think about the elements of A^1 as satisfying x^1, and about the elements of A^0 as satisfying x^0. Then x^1 and x^0 can be both satisfied, but not simultaneously.

To implement the above idea, we extend the relations further, building on the above extension. We suppose without loss of generality that ℓ is a multiple of q', and we set $\ell' = \ell/q'$. Since q' is constant, MONSAT on ℓ' variables is still NP-hard. We take ℓ' pairwise disjoint blocks of size q' of the index set $[\ell]$ and on each block we consider relations $R^0, \ldots, R^h$. We denote by R^i_k the ℓ-ary relation which is obtained by extending R^i from the kth block. Observe that the relations R^i_k are just extensions of elements of $\mathcal{R}$.

After these preparations, we are ready to present the construction. Let $K = \bigwedge_{t=1}^u K_t$ be an instance of E–3SAT in ℓ' variables, with each 3-clause of the form $K_t = x_{i_{t,1}}^{b_t} \vee x_{i_{t,2}}^{b_t} \vee x_{i_{t,3}}^{b_t}$, where $i_{t,1}, i_{t,2}, i_{t,3}$ are indices from $[\ell']$ and b_t is either 0 or 1. Then we map K to the instance $\mathcal{C}$ whose constraints are

$$R^{b_t}_{i_{t,1}} \cup R^{b_t}_{i_{t,2}} \cup R^{b_t}_{i_{t,3}},$$

for each $t \in [u]$, and

$$C_k^j = R_k^j,$$

for each $k \in [\ell']$ and $j \in \{2, \dots, h\}$. This is an instance of E–S since the three relations $R^{b_t}_{i_{t,1}}$, $R^{b_t}_{i_{t,2}}$ and $R^{b_t}_{i_{t,3}}$ are the extensions of the same relation in $\mathcal{R}$. It is quite easy to see that K is satisfiable if and only if $\mathcal{C}$ is satisfiable. Indeed, a satisfying assignment a for the $\mathcal{C}$ can be translated to a satisfying assignment for K by assigning 0 or 1 to x_k according to whether the kth block of a was in A_k^0 or A_k^1 (taking an arbitrary value if it was in none of the two). Similarly, a satisfying assignment b for K can be translated to a satisfying assignment a for $\mathcal{C}$ by picking any element of $A_k^{b_k}$ for the kth block of a. □

An immediate consequence is that under the same conditions H–$S_\emptyset$ is NP-hard too. For an application of this consequence, let LINEQ stand for the CSP in which that alphabet is identified with a finite field F and the ℓ-ary constraints are linear equations over F.

Claim 5.2. H–LINEQ$_\emptyset$ *is* NP*-hard.*

Proof. For each $i \in \ell$, we pick two equations: $x_i = 0$ and $x_i = 1$. Observe that $x_i = 0$ is the same as $\{0\}^i$, the ℓ-ary extension of the unary relation $\{0\}$ on the ith position and we have the same if we replace 0 by 1. By the above observation, the H–CSPs built from relations of these type are NP-hard. □

6. Monotone graph properties

In this section, we consider monotone graph properties in the context of constraint index and relation revealing oracles.

Let us first formulate monotone graph properties as CSPs. Recall that a *monotone graph property* of an n-vertex graph is a monotone Boolean function $\mathcal{P}$ on $\binom{n}{2}$ variables indexed by $\{(i, j), 1 \le i < j \le n\}$, invariant under the induced action of S_n on $[n]$. The goal is to decide, given a graph $G = (V, E)$, whether $E \in \mathcal{P}$.

For a monotone graph property $\mathcal{P}$, we turn it into a CSP problem as follows. Given a graph G, to decide whether G satisfies the property $\mathcal{P}$, we are asked to propose a minimal graph H satisfying $\mathcal{P}$ such that H is a subgraph of G. That is, G is thought of as an instance, and H is an assignment. If H is a subgraph of G then H is considered as satisfying. If not, a violation is given by an edge in H but not in G. That is, the set of constraints in the instance G consists of negations of the variables corresponding to the edges *not* in G. Formally, the CSP $S_\mathcal{P}$ associated with $\mathcal{P}$ has parameters $w = 2$, $q = 1$, $\ell = \binom{n}{2}$, $W_\mathcal{P}$ which consists of the (characteristic vectors of) the graphs (sets of edges), minimal for inclusion satisfying $\mathcal{P}$, and $\mathcal{R} = \{\mathsf{Neg}\}$, where Neg is the negation function. The corresponding constraints are $\mathsf{Neg}(e)$ for every $e \notin E$. A graph $G = (V, E)$ yields an instance consisting of $\mathsf{Neg}(e)$ for $e \notin G$. Then the goal is the same as to decide, given whether there exists an $A \in W_\mathcal{P}$ such that $A \subseteq E$. Notice that the graph property specific part of this model is *fully* left to the specification of the set $W_\mathcal{P}$ of admissible assignments.

Let us examine such CSPs in the trial and error setting. As there is only one relation $\mathcal{R} = \{\mathsf{Neg}\}$, the models revealing or not revealing the violated relation type are equivalent, so naturally the interesting setting is to work with the constraint index and relation revealing oracle. In particular, as already noted in the Introduction, the case when the variable is also revealed is polynomial time equivalent to the "normal" problem. Also note that the union of arity extension and the restricted union of arity extension are the same. Then the arity extension is X–$\mathcal{R} = \{\mathsf{Neg}^e \mid e \in \binom{n}{2}\}$, where $\mathsf{Neg}^e(\alpha_1, \dots, \alpha_{\binom{n}{2}}) = \neg\alpha_e$, and therefore $\bigcup$X–$S_\mathcal{P}$ = $\{\vee_{e \in E'} \mathsf{Neg}^e \mid E' \subseteq \binom{n}{2}\}$ where $\vee$ denotes the or operator. That is, the relations in $\bigcup$X–$S_\mathcal{P}$ are parametrized by sets E' of possible edges where the relation corresponding to E' is equivalent to that at least one edge of the proposed graph is not there in E'.

As a consequence, the $\bigcup$X–$S_\mathcal{P}$ problem becomes the following: Given a graph $G = (V, E)$, and edge sets on n vertices $E_1, \dots, E_m \subseteq \binom{[n]}{2}$, does there exist an $A \in W_\mathcal{P}$ such that $A \subseteq E$ and A excludes at least one edge from each E_i? While every subset of the edge set $\binom{n}{2}$ is in our disposal, in actual applications, the tricky part is to come up with appropriate subsets $E_1, \dots, E_m$, which, together with the minimal instances of the graph property in question, yield that the resulting $\bigcup$X–$S_\mathcal{P}$ problem is hard. This will be the main content in our proofs for the various parts of Theorem 6.1.

This framework naturally extends to directed and bipartite graphs as well as to graphs with one or more designated vertices. Monotone decreasing properties can be treated by replacing Neg with Id, the identity function.

From Theorem 3.3, the complexity of H–$S_{\mathcal{P}\emptyset}$ can be analyzed by considering the complexity of $\bigcup$X–$S_\mathcal{P}$. We do this for the following graph properties (named after the minimal satisfying graphs):

1. Spanning Tree (ST): the property of containing a spanning tree, that is, being connected.
2. Directed Spanning Tree (DST): the property of containing a directed spanning tree rooted at vertex (say) 1, such that all the edges of the spanning tree are directed towards the root.
3. Undirected Cycle Cover (UCC): the property of containing an undirected cycle cover (union of vertex disjoint cycles such that every vertex belongs to some cycle).

4. Directed Cycle Cover (DCC): the property of containing a directed cycle cover (union of vertex disjoint directed cycles such that every vertex belongs to some cycle).
5. Bipartite Perfect Matching (BPM): the property of having a perfect matching in a bipartite graph.
6. Directed Path (DPATH): the property of containing a directed path between two specified vertices s and t.
7. Undirected Path (UPATH): the property of containing an undirected path between two specified vertices s and t.

Connectedness of the given graph and connectivity of two designated vertices, as well as their directed versions, belong to the simplest well-known monotone graph properties that are decidable in polynomial time. Having perfect matchings is perhaps the most famous property in P for bipartite graphs. We show that these problems become NP-hard in the hidden setting. We included the cycle cover problems because we prove hardness of having perfect matchings through hardness of H–DCC$_\emptyset$. The hidden version of having a fixed subgraph, e.g., a clique of constant size is in P because there are only polynomially minimal satisfying graphs and they can be efficiently listed. Unfortunately we are not aware of any monotone property which remains efficiently decidable in the hidden setting for a less trivial reason.

Theorem 6.1. *The following problems are* NP*-hard: (1)* H–ST$_\emptyset$*, (2)* H–DST$_\emptyset$*, (3)* H–UCC$_\emptyset$*, (4)* H–DCC$_\emptyset$*, (5)* H–BPM$_\emptyset$*, (6)* H–DPATH$_\emptyset$*, (7)* H–UPATH$_\emptyset$*.*

Proof. We show that the following problems in the hidden model with the constraint index revealing oracle are NP-hard. In each case, we construct an instance of the $\bigcup$X–S$_\mathcal{P}$ *s.t.* it becomes equivalent to a known NP-hard problem and using Theorem 3.3(b) we can conclude that the hidden version, H–S$_\mathcal{P}$, is NP-hard.

(1) Spanning Tree (H–ST$_\emptyset$) When $\mathcal{P}$ is *connectedness*, $W_\mathcal{P}$ is the set of *Spanning Trees* on n-vertices.
Given $G = (V, E)$, for every vertex $v \in V$, we consider $\binom{n-1}{3}$ edge sets E_{vijk} where

$$E_{vijk} := \{\{v, i\}, \{v, j\}, \{v, k\}\} \quad 1 \le i < j < k \le n.$$

With this choice of E_{vijk}s the $\bigcup$X–ST problem asks if there exists a spanning tree in G which avoids at least one edge from each E_{vijk}. This is equivalent to that every vertex v is incident to at most two edges of the spanning tree A. Spanning trees with this property are just Hamiltonian paths in G.
Thus, the $\bigcup$X–ST problem is equivalent to asking if G contains a Hamiltonian path *i.e.* the HAM–PATH problem in G. Hence, the NP-hard HAM–PATH in G problem reduces to the H–ST$_\emptyset$ problem in G.

(2) Directed Spanning Tree (H–DST$_\emptyset$) Similar to the previous case, $W_\mathcal{P}$ is the set of directed spanning trees rooted at vertex 1.
Let $G = (V, E)$, be a directed planar graph such that the in-degree and the out-degree for every vertex is at most 2. The DHAM–PATH problem in G, *i.e.* determining if there exists directed Hamiltonian path ending at node 1 in G, is NP-hard [5]. Our goal is to reduce the DHAM–PATH problem in G to the H–DST$_\emptyset$ problem in G.
For every vertex $v \in V$, of in-degree 2 we consider the edge sets E_v where $E_v := \{(i, v) \mid (i, v) \in E\}$ with $|E_v| \le 2$ by our choice of G. In addition, for every vertex $v \in V$, of out-degree 2 we consider the edge sets E^v where $E^v := \{(v, i) \mid (v, i) \in E\}$. With these E_vs and E^v, the $\bigcup$X–DST problem asks if there exists a directed spanning tree rooted at vertex 1 that contains at most one edge coming in and at most one edge originated from every vertex. These constraints restrict the directed spanning tree, A to be a DHAM–PATH in G, analogously to the undirected case.
Hence, the NP-hard DHAM–PATH problem reduces to the H–DST$_\emptyset$ problem in G.

(3) Undirected Cycle Cover (H–UCC$_\emptyset$) Here, $W_\mathcal{P}$ is the set of *undirected cycle covers* on n-vertices.
From Hell et al. [6] we know that the problem of deciding whether a graph has a UCC that does not use the cycles of length, (say) 5 is NP-hard. We construct an equivalent instance of $\bigcup$X–UCC as follows. We choose the edge sets $E_C := \{e \mid e \in C\}$ ranging over every length 5 cycle C in G. Then, a UCC satisfying the above conditions cannot contain any 5-cycles.
Hence, an NP-hard problem reduces to the H–UCC$_\emptyset$ problem in G.

(4) Directed Cycle Cover (H–DCC$_\emptyset$) In this case, $W_\mathcal{P}$ is the set of *directed cycle covers* on n-vertices.
The proof follows similar to the undirected case. The NP-hard problem we are interested in is determining if a graph has a DCC that does not use cycles of length 1 and 2 [5]. This problem can be expressed as $\bigcup$X–DCC by choosing the edge sets $E_C := \{e \mid e \in C\}$ for every length 1 and length 2 cycle C in G.

(5) Bipartite Perfect Matching (H–BPM$_\emptyset$) Here, $W_\mathcal{P}$ is the set of *perfect matchings* in a complete bipartite graph with n-vertices on each side.
There is a one-to-one correspondence between perfect matchings in a bipartite graph $G = (A \cup B, E)$ with n vertices on each side and the directed cycle covers in a graph $G' = (V', E')$ on n vertices. Every edge $(i, j) \in E'$ corresponds to an undirected edge $\{i_A, j_B\} \in E$. With this correspondence the H–BPM$_\emptyset$ problem in G is equivalent to the H–DCC$_\emptyset$ problem in G'. Thus, from Theorem 6.1(4) the former becomes NP-hard.

(6) Directed Path (H–DPATH$_\emptyset$) We consider $W_\mathcal{P}$ as the set of *directed paths* from s to t.
It is known that given a layout of a directed graph on a plane possibly containing crossings, the problem of deciding whether there is a crossing-free path from s to t is NP-hard [9]. This condition can be expressed by picking the each edge set E_i as the set of pairs of edges that cross.

(7) Undirected Path (H–UPATH$_\emptyset$) In this case, $W_{\mathcal{P}}$ is the set of *undirected paths* from s to t.
We can apply the same proof method as the one used for the H–DPATH$_\emptyset$ problem on an undirected graph.

This completes the proof for Theorem 6.1. □

7. Hidden CSPs with promise on instances

In this section we consider an extension of the H–CSP framework where the instances satisfy some property. For the sake of simplicity, we develop this subject only for the constraint index revealing model. Formally, let S be a CSP, and let PROM be a subset of all instances. Then S with *promise* PROM is the CSP $\mathrm{S}^{\mathrm{PROM}}$ whose instances are only elements of PROM. One such property is *repetition freeness* where the constraints of an instance are pairwise distinct. We denote by RF the subset of instances satisfying this property. For example $\mathrm{1SAT}^{\mathrm{RF}}$, (as well as H–$\mathrm{1SAT}^{\mathrm{RF}}$) consists of pairwise distinct literals. Such a requirement is quite natural in the context of certain graph problems where the constraints are inclusion (or non-inclusion) of possible edges. The promise H–CSPs framework could also be suitable for discussing certain graph problems on special classes of graphs (e.g., connected graphs, planar graphs, etc.).

We would like to prove an analogue of the transfer theorem with promise. Let us be given a promise PROM for the CSP S of type $\mathcal{R} = \{R_1, \ldots, R_s\}$. The corresponding promise $\bigcup$PROM for $\bigcup$S is defined quite naturally as follows. We say that an instance $\mathcal{C} = (C_1, \ldots, C_m)$ of S, where $C_j = R_{k_j}(x_{j_1}, \ldots, x_{j_q})$, is *included* in an instance $\mathcal{C}' = (C'_1, \ldots, C'_m)$ of $\bigcup$S if for every $j = 1, \ldots, m$ $C'_j = Q_j(x_{j_1}, \ldots, x_{j_q})$ for $Q_j \in \bigcup\mathcal{R}$ such that $R_{k_j} \subseteq Q_j$. Then $\bigcup$PROM is defined as the set of instances in $\mathcal{C}' \in \bigcup$S which include $\mathcal{C} \in$ PROM. In order for the transfer theorem to work, we relax the notion of a solution. A *solution under promise* for $\mathcal{C}' \in \bigcup$PROM has to satisfy two criteria: it is a satisfying assignment when $\mathcal{C}'$ includes a satisfiable instance $\mathcal{C} \in$ PROM, and it is EXCEPTION when $\mathcal{C}'$ is unsatisfiable. However, when all the instances $\mathcal{C} \in$ PROM included in $\mathcal{C}'$ are unsatisfiable but $\mathcal{C}'$ is still satisfiable, it can be either a satisfying assignment or EXCEPTION. We say that an algorithm *solves* $\bigcup \mathrm{S}^{\bigcup \mathrm{PROM}}$ *under promise* if $\forall \mathcal{C}' \in \bigcup$PROM, it outputs a solution under promise.

Using the above definition in the transfer theorem's proof allows the algorithm for H–$\mathrm{S}_{\{\mathcal{V}\}}$ to terminate, at any moment of time, with the conclusion NO as soon as it gets enough information about the instance to exclude satisfiability and without making further calls to the revealing oracle. In some ambiguous cases, it can still call the oracle with an assignment which satisfies the $\bigcup$S-instance. Other cases when the satisfiability of a $\bigcup$S-instance with promise implies the existence of a satisfiable promise-included instance lack this ambiguity. With these notions the proof of Theorem 3.1 goes through and we obtain the following.

Theorem 7.1. *Let* $\mathrm{S}^{\mathrm{PROM}}$ *be a promise* CSP. *(a) If* $\bigcup \mathrm{S}^{\bigcup \mathrm{PROM}}$ *is solvable under promise in time* T *then* H–$\mathrm{S}^{\mathrm{PROM}}_{\{\mathcal{V}\}}$ *is solvable in time* $O((T + s \times \mathrm{comp}(\mathcal{R})) \times m \times \min\{\dim(\bigcup\mathcal{R}), |W_q|\})$.
(b) If H–$\mathrm{S}^{\mathrm{PROM}}_{\{\mathcal{V}\}}$ *is solvable in time* T *then* $\bigcup \mathrm{S}^{\bigcup \mathrm{PROM}}$ *is solvable under promise in time* $O(T \times m \times \mathrm{comp}(\bigcup\mathcal{R}))$.

We apply Theorem 7.1 to the following problems: (1) H–$\mathrm{1SAT}^{\mathrm{RF}}_\emptyset$, repetition free H–1SAT; (2) H–$\mathrm{2SAT}^{\mathrm{RF}}_\emptyset$, repetition free H–2SAT; (3) H–$\mathrm{2COL}^{\mathrm{RF}}_\emptyset$, repetition free H–2COL; (4) H–kWEIGHT$^{\mathrm{RF}}_\emptyset$ the repetition free hidden version of the following problem. The problem kWEIGHT decides if a 0–1 string has Hamming weight at least k. Formally, we have $w = 2$, $q = 1$ and $\mathcal{R} = \{\{0\}\}$ and W consists of words of length ℓ having Hamming weight k. An instance of kWEIGHT is a collection $(C_1, \ldots, C_m)$ of constraints of the form $x_{i_j} = 0$ (formally, $C_j = \{0\}^{i_j}$). (The string behind these constraints is b where $b_t = 0$ if and only if $t \in \{i_1, \ldots, i_m\}$.) In a repetition free instance we have $|\{i_1, \ldots, i_m\}| = m$.

Our main motivation for introducing promises on instances is to study the effect of prohibiting repetition of constraints. This requirement potentially makes the hidden problem easier as it will be indeed the case of 1SAT (in the constraint index revealing model). However, it neither helps in the case of 2SAT nor in the case of the graph property bipartiteness (2COL). (We remark that guessing a 2-coloring could be interpreted as covering the complementer graph by two complete subgraphs and hence could also have been discussed in the framework of the previous section. Here we show hardness of the potentially easier, repetition-free version.) Finally, kWEIGHT is a simple question where the impact of repetition-freeness depends on the parameter k. We have the following.

Theorem 7.2. *(a) Repetition free* H–1SAT *with constraint index revealing oracle is easy, that is* H–$\mathrm{1SAT}^{\mathrm{RF}}_\emptyset \in$ P. *(b)* H–kWEIGHT$_\emptyset$ *is* NP*-hard for certain* k*, but* H–kWEIGHT$^{\mathrm{RF}}_\emptyset \in$ P *for every* k. *(c) Repetition free* H–2SAT*, with constraint index revealing oracle, that is,* H–$\mathrm{2SAT}^{\mathrm{RF}}_\emptyset$ *is* NP*-hard. (d) Repetition free* H–2COL*, that is* H–$\mathrm{2COL}^{\mathrm{RF}}_\emptyset$ *is* NP*-hard.*

Proof. We prove each part of the theorem separately:

(a) We consider every literal as its extended ℓ-ary relation where n is the number of variables. This transforms the $\emptyset$-oracle into a $\{\mathcal{V}\}$-oracle. A repetition free instance of $\bigcup$1SAT is $\mathcal{C} = \{C_1, \ldots, C_m\}$, where each C_j is a disjunction of literals from $\{x_1, \overline{x}_1, \ldots, x_\ell, \overline{x}_\ell\}$ such that there exist m distinct literals $z_1, \ldots, z_m$ with z_j from C_j. A conjunction of literals is satisfiable, if for every $i \in [\ell]$, the literals x_i and $\overline{x}_i$ are not both among them. Hence an algorithm which solves

H–1SAT$^{\mathsf{RF}}_{\emptyset}$ under promise can proceed as follows. Using a maximum matching algorithm it selects pairwise different variables $x_{i_1}, \dots, x_{i_m}$ such that x_{i_j} or $\overline{x}_{i_j}$ is in C_j. If such a selection is not possible it returns EXCEPTION. Otherwise it can trivially find a satisfying assignment.

(b) Again, we extend the relations to their ℓ-ary counterparts so that the $\emptyset$-oracle is transformed into a $\{\mathcal{V}\}$-oracle.
An instance of $\bigcup k$WEIGHT is $\mathcal{C}' = \{C'_1, \dots, C'_m\}$, where there exist subsets $S_1, \dots, S_m$ of $[\ell]$ such that the relation for C_j is the set $\{a \in [\![w]\!]^\ell : a_i = 0 \text{ for some } i \in S_j\}$. Finding a satisfying instance of $\bigcup \mathcal{R}$ is therefore equivalent to finding a hitting set (a transversal) of size (at most) $\ell - k$ for the hypergraph $\{S_1, \dots, S_m\}$. This problem is NP-hard for, say, $0.01\ell < k < 0.99\ell$.
A kWEIGHT$^{\mathsf{RF}}$-instance included in an instance of $\bigcup k$WEIGHT$^{\bigcup \mathsf{RF}}$ corresponding to subsets $S_1, \dots, S_m$ consists of constraints $x_{i_j} \neq 0$ for m different indices $i_1, \dots, i_m$ with $i_j \in S_j$. Obviously, such a set of constraints is satisfiable by an element of W if and only if $m \leq \ell - k$. These observations immediately give the following efficient solution under promise for $\bigcup k$WEIGHT$^{\bigcup \mathsf{RF}}$. If $m > \ell - k$ we return EXCEPTION. Otherwise, using a maximum matching algorithm we find m different places $i_1, \dots, i_m$ with $i_j \in S_j$ (which must exist by the promise) and return an assignment from W which can be found in an obvious way.

(c) To work in the framework of a $\{\mathcal{V}\}$-oracle rather than a $\emptyset$-oracle, we consider every clause as its extended n-ary relation where n is the number of variables. This transforms the $\emptyset$-oracle into a $\{\mathcal{V}\}$-oracle. We reduce 3SAT to $\bigcup$2SAT$^{\bigcup \mathsf{RF}}$ as follows. Let $\phi = \bigwedge_{j=1}^{m} C_j$ be a 3–CNF where

$$C_j = x_{j_1}^{b_1} \vee x_{j_2}^{b_2} \vee x_{j_3}^{b_3}.$$

(Here $b_i \in \{0, 1\}$ and x^1 denotes x, x^0 stands for $\overline{x}$.) For each $j = 1, \dots, m$ we introduce a new variable y_j. We will have $2m$ new clauses:

$$C'_j = x_{j_1}^{b_1} \vee x_{j_2}^{b_2} \vee x_{j_3}^{b_3} \vee y_j^0 \text{ and } C''_j = x_{j_1}^{b_1} \vee x_{j_2}^{b_2} \vee x_{j_3}^{b_3} \vee y_j^1$$

for each j. Put $\phi' = \bigwedge_{j=1}^{m} (C'_j \wedge C''_j)$. Then ϕ is satisfiable if and only if ϕ' is satisfiable. In fact, there is a 1 to 2^m correspondence between the assignments satisfying ϕ and those satisfying ϕ': only the values assigned to the first ℓ variables matter. Also, the included constraints $(x_{j_1}^{b_1} \vee y_j^1)$ and $(x_{j_1}^{b_1} \vee y_j^0)$ for all $j = 1, \dots, m$ form a system of $2m$ different 2–CNFs. Furthermore, if ϕ' is satisfied by an assignment then we can select a satisfiable system of $2m$ pairwise distinct sub-constraints: for each j we pick $s \in \{1, 2, 3\}$ such that $x_{j_s}^{b_s}$ is evaluated to 1 and take $(x_{j_s}^{b_s} \vee y_j^1)$ and $(x_{j_s}^{b_s} \vee y_j^0)$ for $j = 1, \dots, m$.

(d) Here the alphabet is $[\![2]\!] = \{0, 1\}$, $q = 2$, $\mathcal{R}$ has one element "$\neq$", that is $\{(1, 0), (0, 1)\}$. An instance of 2COL$^{\mathsf{RF}}$ consists of a set of constraints of the form $x_u \neq x_v$ for m pairwise distinct unordered pairs $\{u, v\}$ from $\{1, \dots, \ell\}$ (corresponding to the edges of a graph). (Here we again work in the context of the extensions of the relation "$\neq$" to arity $\ell = n$.)
An instance of $\bigcup$2COL is a collection $\{C_1, \dots, C_m\}$, where each C_j is a disjunction of constraints of the form $x_u \neq x_v$. In an equivalent view, an instance of $\bigcup$2COL can be described by the collection of edge sets (graphs) $E_1, \dots, E_m$ on vertex set $[n]$ and a satisfying assignment can be described by a coloring $c : \{1, \dots, n\} \to \{0, 1\}$ such that for every j there exists an edge $e_j \in E_j$ with endpoints having different colors. It is clear that if the edge sets $E_1, \dots, E_m$ are disjoint then the instance is repetition free and the solutions under promise coincide with the solutions in the normal sense.
Let $E_1, \dots, E_m$ be edge sets describing an instance of $\bigcup$2COL. Put $s_j = |E_j|$. For each j we introduce $2s_j$ new vertices: $uvj1, uvj2$ for each $\{u, v\} \in E_j$, $2s_j$ new one-element edge sets $E_{uvj1} = \{\{u, uvj1\}\}$ and $E_{uvj2} = \{\{v, uvj2\}\}$; while E_j is replaced with an edge E'_j set consisting of s_j edges: $\{uvj1, uvj2\}$ for each $\{u, v\} \in E_j$. It turns out that the $\bigcup$2COL problem on the $n + 2\sum_{j=1}^{m} s_j$ vertices with the new $m + 2\sum_{j=1}^{m} s_j$ edge sets is equivalent to the original one and solutions of the two problems can be easily (and efficiently) mapped to each other. The new edge sets are pairwise disjoint and hence the repetition free version of the new $\bigcup$2COL problem is the same as the non-promise version.
Theorem 5.1 shows that non-promise $\bigcup$2COL is NP-hard. By the reduction above, so is its repetition free version. □

On group isomorphism Isomorphism of a hidden multiplication table with a given group, a problem discussed in [3], can also be cast in the framework of promise H–CSPs. We consider the following problem GROUPEQ (equality with a group from a class). Let $\mathcal{G}$ be a family of groups on the set $[k]$, that is, a set of multiplication tables on $[k]$ such that each multiplication table defines a group. The task is to decide whether a hidden group structure $b(\,,\,)$ is equal to some $a(\,,\,)$ from $\mathcal{G}$ and if yes, find such an $a(\,,\,)$. (Note that a solution of the latter task will give the whole table for $b(\,,\,)$.)

We define GROUPEQ($\mathcal{G}$) as a promise CSP as follows. First we consider the CSP ENTRIES($\mathcal{G}$) with the following parameters and type. We have $w = k$, $W = \mathcal{G}$, $\mathcal{R} = \{\{w\} : w \in [k]\}$, $\ell = k^2$. It will be convenient to consider assignments as $k \times k$ tables with entries from $[k]$, that is, functions $[k]^2 \to [k]$. (Implicitly, we use a bijection between the index set $\{1, \dots, \ell\}$ and $[k]^2$.) The number of constraints is $m = k^2$ and an instance is a collection $x_{(u_h, v_h)} = b_h$, where $h = 1, \dots, m$, and $u_h, v_h, b_h \in [k]$. Thus the assignment satisfying ENTRIES($\mathcal{G}$) are $k \times k$ multiplication tables from $\mathcal{G}$ which have prescribed values at k^2 (not necessarily distinct) places.

We say that an instance for ENTRIES($\mathcal{G}$) belongs to the promise GROUP if two conditions are satisfied. Firstly, there is one constraint for the value taken by each place. Formally, the map $\tau : h \mapsto (u_h, v_h)$ is a bijection between $\{1, \dots, m\}$

and $[k]^2$. As a consequence, by setting $b(u,v) := b_{\tau^{-1}(u,v)}$, we have a constraint $x_{u,v} = b(u,v)$ for pair $(u,v) \in [k]^2$. The second – essential – condition is that the multiplication given by $b(\,,\,)$ defines a group structure on $[k]$. The promise problem GROUPEQ($\mathcal{G}$) is the problem ENTRIES($\mathcal{G}$)$^{\mathsf{GROUP}}$.

We consider the promise problem H–ENTRIES($\mathcal{G}$)$^{\mathsf{GROUP}}_{\{V\}}$ (which we denote by H–GROUPEQ($\mathcal{G}$) for short) and the corresponding problem $\bigcup$ENTRIES($\mathcal{G}$)$^{\bigcup\mathsf{GROUP}}_{\{V\}}$ (short notation: $\bigcup$GROUPEQ($\mathcal{G}$)). In this H–CSP, if $a(\,,\,)$ is different from $b(\,,\,)$, the oracle reveals a pair (u,v) such that $a(u,v) \neq b(u,v)$.

We note here that the case of H–GROUPEQ($\mathcal{G}$) where $\mathcal{G}$ consists of all isomorphic copies of a group G in fact covers the problem of finding an isomorphism with G discussed in [3]. For that problem, the input to the verification oracle is a bijection $\phi : [k] \to G$. Recall that $b(\,,\,)$ encodes the hidden group structure, and we assume G is specified by the binary relation $g(\,,\,)$. Then, in the case when ϕ is not an isomorphism, the oracle has to reveal $u, v \in [k]$ such that, the product $g(\phi(u), \phi(v))$ does not equal $\phi(b(u,v))$ in G. Indeed, given ϕ we can define (and even compute) the multiplication $a_\phi(\,,\,)$ on $[k]$ – by taking $a_\phi(x,y) = \phi^{-1}(g(\phi(x), \phi(y))$ – such that ϕ becomes an isomorphism from the group given by $a_\phi(\,,\,)$ to G. Then ϕ is an isomorphism from the group given by $b(\,,\,)$ if and only if $a_\phi(\,,\,) = b(\,,\,)$. Furthermore, if it is not the case then the oracle given in [3] reveals a pair (u,v) such that $a_\phi(u,v) = b(u,v)$, exactly what is expected from a revealing oracle for H–GROUPEQ($\mathcal{G}$).

An instance of $\bigcup$ENTRIES($\mathcal{G}$) consists of k^2 constraints expressing that $a(u_h, v_h) \in S_h$ where $S_h \in 2^{[k]} \setminus \emptyset$ for $h \in [m] = [k^2]$. An instance of the promise version $\bigcup$GROUPEQ($\mathcal{G}$) (which is equal to $\bigcup$ENTRIES($\mathcal{G}$)$^{\bigcup\mathsf{GROUP}}_{\{V\}}$) should satisfy that $\{(u_h, v_h) : h = 1, \ldots, m\} = [k]^2$, that is, our constraints are actually $x_{(u,v)} \in S(u,v)$ for a map $S(\,,\,) : [k]^2 \to 2^{[k]}$. Furthermore, there is a map $b(\,,\,) : [k]^2 \to [k]$ with $b(u,v) \in S(u,v)$ for every $(u,v) \in [k]^2$ such that $b(\,,\,)$ gives a group structure.

Now we are ready to reprove Theorem 11 in [3]. Note that our proof is considerably shorter than the original proof.

Theorem 7.3. *The problem* H–GROUPEQ($\mathcal{G}$) *is* NP*-hard.*

Proof. Let p be a prime. We show that finding Hamiltonian cycles in Hamiltonian digraphs of size p is reducible in polynomial time to H–GROUPEQ($\mathcal{G}$). The former problem is NP-hard, since an algorithm that in polynomial time finds a Hamiltonian cycle in a Hamiltonian digraph obviously can decide if an arbitrary digraph G has a Hamiltonian cycle: it just runs on G and then tests if the outcome is indeed a Hamiltonian cycle.

Choose $\mathcal{G}$ as the set of all group structures on $[p]$. As every group having p elements is isomorphic to Z_p, $\mathcal{G}$ coincides with the group structures on $[p]$ isomorphic to Z_p. We essentially translate the arguments given in [3] to the setting of $\bigcup$GROUPEQ as follows. This suffices to prove the statement due to the transfer theorem stated in Theorem 7.1.

Let $([p], E)$ be a Hamiltonian directed graph (without loops) on $[p]$. Fix $z \in [p]$. For $u \in [p]$, let $S(u,z) = \{v : (u,v) \in E\}$, and $S(u,v) = [p]$ for $v \neq z$. Let $\phi : [p] \to \{0, \ldots, p-1\} = Z_p$ be a bijection that defines a Hamiltonian cycle in $([p], E)$. Then $b(x,y) = a_\phi(x,y) := \phi^{-1}(\phi(x) + \phi(y))$ gives a group structure on $[p]$ (isomorphic to Z_p via ϕ) consistent with the constraints given by $S(\,,\,)$. Conversely, if $b(\,,\,)$ gives a group structure (necessarily isomorphic to Z_p) consistent with $S(\,,\,)$ then the pairs $(u, b(u,z))$ $(u \in [p])$ form a Hamiltonian cycle in $([p], E)$. Thus finding Hamiltonian cycles in Hamiltonian digraphs on p points can be reduced to $\bigcup$GROUPEQ on p elements. □

As an example, suppose we have $p = 3$, and the edges are $2 \to 1, 1 \to 3, 3 \to 2$. Then set $z = 2$, so $\phi(2) = 1$, $\phi(1) = 2$, $\phi(3) = 0$. (That is, i is the $\phi(i)$th vertex to be visited in this Hamiltonian cycle, where $\phi(i)$ should be understood as modulo p.) It can be verified that $b(x,2) \in S(x,2) = \{y : (x,y) \in E\}$. On the other hand, if we set $b(x,y)$ to be isomorphic to Z_p by the correspondence just given by ϕ, then the path $(u, b(u,2))$ forms a Hamiltonian cycle.

Acknowledgments

The authors are grateful to the anonymous reviewers for their helpful remarks and suggestions. This research was supported in part by the Hungarian National Research, Development and Innovation Office – NKFIH (Grant K115288), the Singapore Ministry of Education and the National Research Foundation, also through the Tier 3 Grant "Random numbers from quantum processes" MOE2012-T3-1-009, by the European Commission IST STREP project Quantum Algorithms (QALGO) 600700, the French ANR Blanc Program under contract ANR-12-BS02-005 (RDAM project), and Australian Research Council DECRA DE150100720.

References

[1] Xiaohui Bei, Ning Chen, Liyu Dou, Xiangru Huang, Ruixin Qiang, Trial and error in influential social networks, in: Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, Ramasamy Uthurusamy (Eds.), KDD, ACM, 2013, pp. 1016–1024.
[2] Xiaohui Bei, Ning Chen, Shengyu Zhang, On the complexity of trial and error, CoRR, 2012, arXiv:1205.1183 [abs].
[3] Xiaohui Bei, Ning Chen, Shengyu Zhang, On the complexity of trial and error, in: Dan Boneh, Tim Rougarden, Joan Feigenbaum (Eds.), STOC, ACM, 2013, pp. 31–40.
[4] Xiaohui Bei, Ning Chen, Shengyu Zhang, Solving linear programming with constraints unknown, in: Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, Bettina Speckmann (Eds.), ICALP, LNCS, 2015, pp. 129–142.
[5] M.R. Garey, David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[6] Pavol Hell, David G. Kirkpatrick, Jan Kratochvíl, Igor Kríz, On restricted two-factors, SIAM J. Discrete Math. 1 (4) (1988) 472–484.
[7] Gábor Ivanyos, Raghav Kulkarni, Youming Qiao, Miklos Santha, Aarthi Sundaram, On the complexity of trial and error for constraint satisfaction problems, in: Automata, Languages, and Programming – Proceedings of the 41st International Colloquium, Part I, ICALP 2014, Copenhagen, Denmark, July 8–11, 2014, 2014, pp. 663–675.
[8] Subhash Khot, On the power of unique 2-prover 1-round games, in: Proceedings on 34th Annual ACM Symposium on Theory of Computing, Montréal, Québec, Canada, May 19–21, 2002, 2002, pp. 767–775.
[9] Jan Kratochvíl, Anna Lubiw, Jaroslav Nešetřil, Noncrossing subgraphs in topological layouts, SIAM J. Discrete Math. 4 (2) (March 1991) 223–244.
[10] Thomas J. Schaefer, The complexity of satisfiability problems, in: Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, Alfred V. Aho (Eds.), STOC, ACM, 1978, pp. 216–226.