Dichotomy for Digraph Homomorphism Problems

Tomás Feder* Jeff Kinne † Arash Rafiey ‡

Abstract

We consider the problem of finding a homomorphism from an input digraph G to a fixed digraph H. We show that if H admits a weak-near-unanimity polymorphism ϕ then deciding whether G admits a homomorphism to H (HOM(H)) is polynomial time solvable. This confirms the conjecture of Maroti and McKenzie [MM08], and consequently implies the validity of the dichotomy conjecture due to Feder and Vardi [FV93].

1 Introduction

For a digraph G, let $V(G)$ denote the vertex set of G and let $A(G)$ denote the arcs (aka edges) of G. An arc (u, v) is often written as simply uv to shorten expressions. Let $|G|$ denote the number of vertices in G.

A *homomorphism* of a digraph G to a digraph H is a mapping g of the vertex set of G to the vertex set of H so that for every arc uv of G the image $g(u)g(v)$ is an arc of H. A natural decision problem is whether for given digraphs G and H there is a homomorphism of G to H. If we view (undirected) graphs as digraphs in which each edge is replaced by the two opposite directed arcs, we may apply the definition to graphs as well. An easy reduction from the k-coloring problem shows that this decision problem is NP-hard: a graph G admits a 3-coloring if and only if there is a homomorphism from G to K_3, the complete graph on 3 vertices. As a homomorphism is easily verified if the mapping is given, the homomorphism problem is contained in NP and is thus NP-complete.

*268 Waverley Street, Palo Alto, CA 94301, United States, tomas@theory.stanford.edu
†Indiana State University, IN, USA, jkinne@cs.indstate.edu
‡Indiana State University, IN, USA arash.rafiey@indstate.edu and Simon Fraser University, BC, Canada, arashr@sfu.ca

The following version of the problem has attracted much recent attention. For a fixed digraph H the problem $HOM(H)$ asks if a given input digraph G admits a homomorphism to H. Note that while the above reduction shows $HOM(K_3)$ is NP-complete, $HOM(H)$ can be easy (in P) for some graphs H: for instance if H contains a vertex with a self-loop, then every graph G admits a homomorphism to H. Less trivially, for $H = K_2$ (or more generally, for any bipartite graph H), there is a homomorphism from G to K_2 if and only if G is bipartite. A very natural goal is to identify precisely for which digraphs H the problem $HOM(H)$ is easy. In the special case of graphs the classification has turned out to be this: if H contains a vertex with a self-loop or is bipartite, then $HOM(H)$ is in P, otherwise it is NP-complete [HN90] (see [B05, S10] for shorter proofs). This classification result implies a *dichotomy* of possibilities for the problems $HOM(H)$ when H is a graph, each problem being NP-complete or in P. However, the dichotomy of $HOM(H)$ remained open for general digraphs H. It was observed by Feder and Vardi [FV93] that this problem is equivalent to the dichotomy of a much larger class of problems in NP, in which H is a fixed finite relational structure. These problems can be viewed as *constraint satisfaction problems* with a fixed template H [FV93], written as $CSP(H)$. A constraint satisfaction problem $CSP(H)$ consists of (a) a relational structure H that specifies a set V of variables that each come from some domain D and (b) a set C of constraints giving restrictions on the values allowed on the variables.

The question is whether all constraints can be simultaneously satisfied. 3SAT is a prototypical instance of CSP, where each variable takes values of *true* or *false* (a domain size of two) and the clauses are the constraints. Digraph homomorphism problems can also easily be converted into CSPs: the variables V are the vertices of G, each must be assigned a vertex in H (meaning a domain size of $|V(H)|$), and the constraints encode that each arc of G must be mapped to an arc in H.

Feder and Vardi argued in [FV93] that in a well defined sense the class of problems $CSP(H)$ would be the largest subclass of NP in which a dichotomy holds. A fundamental result of Ladner [L75] asserts that if $P \neq NP$ then there exist NP-intermediate problems (problems neither in P nor NP-complete), which implies that there is no such dichotomy theorem for the class of *all* NP problems. Non-trivial and natural subclasses which do have dichotomy theorems are of great interest. Feder and Vardi made the following *Dichotomy Conjecture*: every problem $CSP(H)$ is NP-complete or is in P. This problem has animated much research in theoretical computer science. For instance the conjecture has been verified when H is a conservative relational structure [B11], or a digraph with all in-degrees and all-out-degrees at least one [BKN09]. Numerous special cases of this conjecture have been verified [ABISV09, B06, BH90, BHM88, CVK10, D00, F01, F06, FMS04, LZ03, S78].

Bulatov announced his proof for the conjecture earlier this year [B17] and later Zhuk [Z17] also announced another algebraic proof of the conjecture.

It should be remarked that constraint satisfaction problems encompass many well known computational problems, in scheduling, planning, database, artificial intelligence, and constitute an important area of applications, in addition to their interest in theoretical computer

science [CKS01, D92, V00, K92].

While the paper of Feder and Vardi [FV93] did identify some likely candidates for the boundary between easy and hard CSP-s, it was the development of algebraic techniques by Jeavons [J98] that lead to the first proposed classification [BJK05]. The algebraic approach depends on the observation that the complexity of $CSP(H)$ only depends on certain symmetries of H, the so-called *polymorphisms* of H. For a digraph H a polymorphism ϕ of arity k on H is a homomorphism from H^k to H. Here H^k is a digraph with vertex set $\{(a_1, a_2, \ldots, a_k) | a_1, a_2, \ldots, a_k \in V(H)\}$ and arc set $\{(a_1, a_2, \ldots, a_k)(b_1, b_2, \ldots, b_k) \mid a_i b_i \in A(H) \text{ for all } 1 \leq i \leq k\}$. For a polymorphism ϕ, $\phi(a_1, a_2, \ldots, a_k)\phi(b_1, b_2, \ldots, b_k)$ is an arc of H whenever $(a_1, a_2, \ldots, a_k)(b_1, b_2, \ldots, b_k)$ is an arc of H^k.

Over time, one concrete classification has emerged as the likely candidate for the dichotomy. It is expressible in many equivalent ways, including the first one proposed in [BJK05]. There were thus a number of equivalent conditions on H that were postulated to describe which problems $CSP(H)$ are in P. For each, it was shown that if the condition is not satisfied then the problem $CSP(H)$ is NP-complete (see also the survey [HN08]). One such condition is the existence of a weak near-unanimity polymorphism (Maroti and McKenzie [MM08]). A polymorphism ϕ of H of arity k is a k *near unanimity function* (k-nu) on H, if $\phi(a, a, \ldots, a) = a$ for every $a \in V(H)$, and $\phi(a, a, \ldots, a, b) = \phi(a, a, \ldots, b, a) = \cdots = \phi(b, a, \ldots, a) = a$ for every $a, b \in V(H)$. If we only have $\phi(a, a, \ldots, a) = a$ for every $a \in V(H)$ and $\phi(a, a, \ldots, a, b) = \phi(a, a, \ldots, b, a) = \cdots = \phi(b, a, \ldots, a)$ [not necessarily a] for every $a, b \in V(H)$, then ϕ is a *weak k-near unanimity function* (weak k-nu).

Given the NP-completeness proofs that are known, the proof of the Dichotomy Conjecture reduces to the claim that a relational structure H which admits a weak near-unanimity polymorphism has a polynomial time algorithm for $CSP(H)$. As mentioned earlier, Feder and Vardi have shown that is suffices to prove this for $HOM(H)$ when H is a digraph. This is the main result of our paper.

Note that the real difficulty in the proof of the *graph* dichotomy theorem in [HN90] lies in proving the NP-completeness. By contrast, in the *digraph* dichotomy theorem proved here it is the polynomial-time algorithm that has proven more difficult.

While the main approach in attacking the conjecture has mostly been to use the highly developed techniques from logic and algebra, and to obtain an algebraic proof, we go in the opposite direction and develop a combinatorial algorithm. Our main result is the following.

Theorem 1.1 *Let H be a digraph that admits a weak near-unanimity function. Then $HOM(H)$ is in P. Deciding whether an input digraph G admits a homomorphism to H can be done in time $\mathcal{O}(|G|^6 |H|^{k+3})$.*

Together with the NP-completeness result of [MM08], this settles the CSP Conjecture in the affirmative. We note that H is fixed and also $k \leq 2|H|$ according to [JB17].

Our Methods, Very High Level View We start with a general digraph H and a weak k-nu ϕ of H. We turn the problem $HOM(H)$ into a related problem of seeking a homomorphism with lists of allowed images. The *list homomorphism problem* for a fixed digraph H, denoted $LHOM(H)$, has as input a digraph G, and for each vertex x of G an associated list (set) of vertices $L(x) \subseteq V(H)$, and asks whether there is a homomorphism g of G to H such that for each $x \in V(G)$, the image $g(x)$ is in $L(x)$. Such a homomorphism is called a *list homomorphism* of G to H with respect to the lists L. List homomorphism problems are known to have nice dichotomies. For instance when H is a reflexive graph (each vertex has a loop), the problem $LHOM(H)$ is polynomial when H is an interval graph and is NP-complete otherwise [FH98]. Similar list homomorphism dichotomies were proved for general graphs [FHH03, FHH07], and more recently also for digraphs [HR11]. In fact, motivated by the results in [FH98, FHH03], Bulatov [B11] proved that the list version of constraint satisfaction problems has a dichotomy for general relational systems.

It is not difficult to see that there are digraphs H such that $HOM(H)$ is polynomial while $LHOM(H)$ is NP-complete. For instance, the reflexive four-cycle H has loops and so $HOM(H)$ is trivial, while $LHOM(H)$ is NP-complete since H is not an interval graph. However, we transform the problem $HOM(H)$ into a restricted version of $LHOM(H)$ in which the lists satisfy an additional property related to the weak k-nu ϕ.

One of the common ingredients in CSP algorithms is the use of consistency checks to reduce the set of possible values for each variable (see, for example the algorithm outlined in [HN04] for $CSP(H)$ when H admits a near unanimity function). Our algorithm includes such a consistency check as a first step. We begin by performing a pair consistency check of the list of vertices in the input digraph G. For each pair (x, y) of $V(G) \times V(G)$ we consider a list of possible pairs (a, b), $a \in L(x)$ (the list in H associated with $x \in G$) and $b \in L(y)$. Note that if xy is an arc of G and ab is not an arc of H then we remove (a, b) from the list of (x, y). Moreover, if $(a, b) \in L(x, y)$ and there exists z such that there is no c for which $(a, c) \in L(x, z)$ and $(c, b) \in L(z, y)$ then we remove (a, b) from the list of (x, y). We continue this process until no list can be modified. If there are empty lists then clearly there is no list homomorphism.

After performing pair consistency checks (and repeating the consistency checks throughout the algorithm), the main structure of the algorithm is to perform *pairwise elimination*, which focuses on two vertices a, b of H that occur together in some list $L(x), x \in V(G)$, and finds a way to eliminate a or b from $L(x)$ without changing a feasible problem into an unfeasible one. In other words if there was a list homomorphism with respect to the old lists L, there will still be one with respect to the updated lists L. This process continues until either a list becomes empty, certifying that there is no homomorphism with respect to L (and hence no homomorphism at all), or until all lists become singletons, specifying a concrete homomorphism of G to H. This technique, due to the last author, has been successfully used in several other papers [HR11, HR12, EHLR14]. In this paper, the choice of which a or b is eliminated, and how, is governed by the given weak near-unanimity polymorphism ϕ. In fact, we define a family of mappings $f_x, x \in V(G)$ which are each polymorphisms

derived from ϕ and use these polymorphisms as a guide. The heart of the algorithm is a delicate procedure for updating the lists $L(x)$ and polymorphisms f_x in such a way that (i) feasibility is maintained, and (ii) the polymorphisms f_x remain polymorphisms (which is key to maintaining feasibility).

2 Issue with the previous manuscript

In this brief update we aim the discussion at those who already are familiar with our approach from reading this or another manuscript. Our algorithm makes a decision based on the output of a test $T_{x,a,b}$ on a smaller instance of the homomorphism problem. Here a, b are possible images for $x \in V(G)$, of a homomorphism from G to H.

The present manuscript assumes that the test $T_{x,a,b}$ outputs"yes" and based on the correctness of the test a is removed from further consideration for x. The test $T_{x,a,b}$ uses the properties of the weak-nuf polymorphism ϕ. However, it is conceivable that the test $T_{x,a,b}$ fails and this means we should not remove a from the list of possible images of x. We had incorrectly claimed in the manuscript that the properties of ϕ and pre-tests in the algorithm guarantee the test always passes. But we can simply construct such an example where the test must fail in the algorithm as follows. Let H be a digraph with two weakly connected components H_1, H_2. The weak-nu polymorphism ϕ could be of arity 3 and such that for every $a \in H_1$ and every $b \in H_2$, $\phi(a, b, b) = \phi(b, b, a) = \phi(b, a, b) = c$ for some $c \in H_2$. Suppose there exists a homomorphism from G (weakly connected) to H that maps x to a and hence the entire graph G must be mapped to H_1. Moreover, suppose there is no homomorphism from G to H_2. The algorithm does consider the test $T_{x,a,b}$ eventually for such G and H. According to the test $T_{x,a,b}$, we remove a from further consideration for x which leads us to remove the possible homomorphism from G to H.

Note that one can assume H is weakly connected as follows. Suppose H_1, H_2 are balanced digraphs with ℓ levels (we can partitioned the vertics of H_i, $i = 1, 2$, into ℓ parts where all the arcs of H_i go from a vertex in some part j to part $j+1$). Then an extra vertex a' can be added and connected to all the vertices of H_1, H_2 on the lowest level. This way we obtain the weakly connected digraph $H = H_1 \cup H_2 \cup \{a'\}$ with $\ell + 1$ levels.

We may assume G is also balanced and has ℓ levels. An extra vertex x' can be added to G with arcs to every vertex of G on the lowest level. Now $G' = G \cup \{x'\}$ is also a balanced digraph with $\ell+1$ levels. Note that in any homomorphism from G' to H, x' must be mapped to a' and any other vertex of G' must map to $H - \{x\}$.

We note that Ross Willard may post a concrete counter-example (and further discussion of his example) for which H contains 197 vertices in H. The example is inspired from instances of the CSP problem, so called Semi-lattice block Maltsev.

3 Algorithm

An *oriented walk (path)* is obtained from a walk (path) by orienting each of its edges. The *net-length* of a walk W, is the number of forward arcs minus the number of backward arcs following W from the beginning to the end. An *oriented cycle* is obtained from a cycle by orienting each of its edges. We say two oriented walks X, Y are congruent if they follow the same patterns of forward and backward arcs.

Given digraphs G and H, let $G \times H^k$ be a digraph on the vertices $\{(y; a_1, a_2, \ldots, a_k) | y \in V(G), a_i \in V(H), 1 \leq i \leq k\}$ with the arcs $(y; a_1, a_2, ..., a_k)(y'; b_1, b_2, ..., b_k)$ where yy' is an arc of G and each $a_i b_i$, $1 \leq i \leq k$, is an arc of H. By convention, we shall further restrict the use of the symbol $G \times H^k$ to the digraph induced on the vertices $\{(y; a_1, a_2, \ldots, a_k) | y \in V(G), a_i \in L(y), 1 \leq i \leq k\}$ where $L(y)$ is the set of vertices in H that are being considered as images of a homomorphism from G to H.

Definition 3.1 (Homomorphism consistent with Lists) *Let G and H be digraphs. For each $x \in V(G)$, let list of x, $L(x)$, be a subset of H. Let $k > 1$ be a constant integer.*

A function $f : G \times H^k \to H$ is a homomorphism consistent with L *if the following hold.*

- List property : *for every $x \in V(G)$ and every $a_1, a_2, \ldots, a_k \in L(x)$, $f(x; a_1, a_2, \ldots, a_k) \in L(x)$*
- Adjacency property: *for every $x, y \in V(G)$ and every $a_1, ..., a_k \in L(x), b_1, ..., b_k \in L(y)$, if xy is an arc of G and $a_i b_i$ is an arc of H for each $1 \leq i \leq k$ then $f(x; a_1, ..., a_k) f(y; b_1, ..., b_k)$ is an arc of H.*

In addition if f has the following property then we say f has the *weak-nu property.*

- for every $x \in V(G), \{a, b\} \subseteq L(x)$, we have $f(x; a, b, b, ..., b) = f(x; b, a, b, ..., b) = ... = f(x; b, b, b, ...a)$.

We note that this definition is tailored to our purposes and in particular differs from the standard definition of weak k-nu as follows.

(a) f is based on two digraphs G and H rather than just H (we think of this as starting with a traditional weak k-nu on H and then allowing it to vary somewhat for each $x \in G$),

(b) We do not require that $f(y; d, d, d, ..., d) = d$ (this is not required in our algorithm, and in fact is more convenient to leave out).

Notation For simplicity let $(b^k, a) = (b, b, \ldots, b, a)$ be a k-tuple of all b's but with an a in the k^{th} coordinate. Let $(x; b^k, a)$ be a $(k+1)$-tuple of x, $(k-1)$ b's and a in the $(k+1)^{th}$ coordinate.

Algorithm 1 The main algorithm for solving the digraph homomorphism problem.

1: **function** DIGRAPHHOM(G, H, ϕ) ▷ G and H digraphs, ϕ a weak k-nu on H
2: **for all** $x \in G$, let $L(x) = V(H)$
3: **for all** $x \in G$ and $a_1, ..., a_k \in V(H)$, let $f(x; a_1, ..., a_k) = \phi(a_1, ..., a_k)$
4: **if** PREPROCESSING(G, H, L) is false **then** print "no homomorphism" and terminate
5: REMOVENOTMINORITY(G, H, L, f)
6: Note: now, for all $x \in V(G)$ and $a, b \in L(x)$ we know $f(x; b^k, a) = a$
7: REMOVEMINORITY(G, H, L, f)
8: **if** RemoveMinority produces a homomorphism **then return** true
9: **else return** false

3.1 Main Procedure

The main algorithm starts with applying the arc consistency and pair consistency on the lists L by calling Algorithm 2.

Algorithm 4 (RemoveNotMinority function) is the key subroutine of the main algorithm. It starts with $w = (x; b^k, a)$ where $f(w) = c \neq a$ and then its goal is to modify f by setting $f(x; e_1, e_2, \ldots, e_k) = f(w)$ for every k-tuple $e_1, e_2, \ldots, e_k \in L(x)$ with $f(x; e_1, e_2, \ldots, e_k) = a$. Now in order to do this modification and have a homomorphism from $G \times H^k$ to H consistent with L, it recursively solves a smaller instance of the problem and if the test is successful, then it modifies f.

After the main loop in Algorithm 1, we end up with a so-called Maltsev or minority instance of the problem – in which we have a homomorphism f consistent with L such that for every $y \in V(G)$ and every $c, d \in L(y)$ we have $f(y; c^k, d) = d$. We argue in the next subsection that such instances can be solved by using the known algorithm of [BD06] (see the remark at the end of Subsection 3.2). The Maltsev/minority instances can also be solved in a manner similar to our arguments for RemoveNotMinority (see Section 5) .

In what follows we give an insight of why the weak-nu property of H is necessary for our algorithm. For contrary suppose $w_1 = (x; b^k, a)$ with $f(w_1) = c$ and $w_2 = (x; a, b, b, \ldots, b)$ with $f(w_2) = d$. If $d = a$ then in RemoveNotMinority we try to remove a from $L(x)$ if we start with w_1 while we do need to keep a in $L(x)$ because we later need a in $L(x)$ for the Maltsev algorithm. It might be the case that $d \neq a$ but during the execution of Algorithm 4 for some $w_3 = (x; b^k, e)$ with $f(w_3) \neq e$ we set $f(w_3)$ to e. So we need to have $f(w_1) = f(w_2)$, the weak-nu property, to start in the main algorithm, Algorithm 1.

Definition 3.2 (f-closure of a list :) *We say a set $S \subseteq L(y)$ is closed under f if for every k-tuple $a'_1, a'_2, \ldots, a'_k \in S$ we have $f(y; a'_1, a'_2, \ldots, a'_k) \in S$.*

For $S \subseteq L(y)$, let $\widehat{f}_{y,S}$ be a minimal set that includes all the elements of S and it is closed under f.

Definition 3.3 *We say $a \in L(x)$ is* relevant *if $\exists w_2 \in V(G \times H^k)$ such that $f(w_2) = a$.*

Algorithm 2 Update lists of x, y based on edge constraints and pair constraint. Call by reference, i.e. the update to L will be reflected in the calling function

function PREPROCESSING(G, H, L)
  Input: digraphs G, H, lists $L(x) \subseteq V(H)$ for each x
    ▷ L lists are unary and binary
    ▷ The update to L would be available for the function calling PreProcessing
  ARCCONSISTENCY(G, H, L) and PAIRCONSISTENCY(G, H, L)

function ARCCONSISTENCY(G, H, L)
  update=True
  while update **do**
    if $\exists xy(yx) \in A(G), a \in L(x)$ s.t. $\nexists b \in L(y)$ with $ab(ba) \in A(H)$ **then**
      remove a from $L(x)$ and set update=True.
    else update=False.
  if there is an empty list **then return** false.

function PAIRCONSISTENCY(G, H, L)
  for all $(x, y) \in V(G) \times V(G)$ **do** set $L(x, y) = \{(a, b) | a \in L(x), b \in L(y)\}$
  for all $x \in V(G)$ **do** set $L(x, x) = \{(a, a) | a \in L(x)\}$.
  for all $xy \in A(G)$, $a \in L(x), b \in L(y)$ **do**
    if $ab \notin A(H)$ **then** remove (a, b) from $L(x, y)$.
  update=True
  while update **do**
    if $\exists x, y, z$ s.t. $\nexists c \in L(z)$ s.t. $(a, c) \in L(x, z) \& (c, b) \in L(z, y)$ **then**
      remove (a, b) from $L(x, y)$ and set update=True.
    else update=False.
  if there is an empty list **then return** false.

Definition 3.4 (restriction of f to a sub-list:) *Let L' be a subset of L, i.e. $\forall y \in V(G)$, $L'(y) \subseteq L(y)$. Let $f|_{L'}$ denote the restriction of f under L', i.e. for all $y \in V(G)$, and for all $a_1, a_2, \ldots, a_k \in L'(y)$ we have $f|_{L'}(y; a_1, a_2, \ldots, a_k) = f(y; a_1, a_2, \ldots, a_k)$.*

Let $X : x_1, x_2, \ldots, x_n$ be an oriented walk in G. Let $L(X)$ denote the vertices of H that lie in the list of the vertices of X. Let $X[x_i, x_j]$, $1 \leq i \leq j \leq n$, denote the induced sub-path of X from x_i to x_j.

Definition 3.5 (induced bi-clique) *We say two vertices x, y induced a bi-clique B in L if there exist vertices $a_1, a_2, \ldots, a_r \in L(x)$, $r > 1$ and $b_1, b_2, \ldots, b_s \in L(y)$ such that $(a_i, b_j) \in L(x, y)$ for every $1 \leq i \leq r$ and $1 \leq j \leq s$. Let Y be an oriented path from x to y. We say bi-clique B is minimal if for every $a_i, a_j \in L(x)$ of B, (a_i, a_j) does not induced a Bi-clique on x, x_1 where $x_1 \in Y - y$.*

Let $a_1, a_2 \in L(x)$ and suppose there exist $b_1, b_2 \in L(y)$ such that $(a_1, b_1), (a_1, b_2), (a_2, b_1)$, $(a_2, b_2) \in L(x, y)$. Then $\widehat{f}_{x,\{a_1,a_2\}}$ and $\widehat{f}_{y,\{b_1,b_2\}}$ induce a bi-clique on x, y.

Definition 3.6 (weakly connected component in lists L and cut points) *Let $G \times_L H$ be the digraph with vertices $\{(x, a) | x \in V(G), a \in L(x)\}$. The arc set of $G \times_L H$ is $A(G \times_L H) = \{(y, d)(y', d') |\ yy' \in A(G), dd' \in A(H), (d, d') \in L(y, y')\} \cup \{(y, d)(y', d') |\ y'y \in A(G), dd' \in A(H), (d, d') \in L(y, y')\}$. We say (x, a) is a cut vertex if either x is a cut-vertex in G or by removing (x, a) the number of connected component of $G \times_L H$ (after Preprocessing; updating the pair lists) increases. We say a connected component C of $G \times_L H$ is valid if for every $y \in V(G)$, there exists a vertex $b \in L(y)$ such that $(y, b) \in C$.*

Observation 3.7 *If there exists a homomorphism $g : V(G) \to V(H)$ then all the vertices $(y, g(y))$, $y \in V(G)$ belong to the same connected component of $G \times_L H$.*

Definition 3.8 ($G_L(x, a, c)$) *For $x \in V(G)$, and $a, c \in L(x)$, let $G_L(x, a, c)$ be a digraph with vertices $V(G_L(x, a, c)) = \{(y, d, e) | (c, e) \in L(x, y), d \in L(y)\}$. The arc set of $G_L(x, a, c)$ is $A(G_L(x, a, c)) = \{(y, d, e)(y', d', e') |\ yy' \in A(G), dd', ee' \in A(H), ed' \notin A(H)\} \cup$ $\{(y, d, e)(y', d', e') |\ y'y \in A(G), d'd, e'e \in A(H), d'e \notin A(H)\}$. Moreover, every vertex of $G_L(x, a, c)$ is reachable from (x, a, c). Note that (x, a, c) is a vertex of $G_L(x, a, c)$.*

Definition 3.9 *We say $w = (x; b^k, a)$ is a non-minority pair if $f(w) = c \neq a$.*

Algorithm 3 It prepares the arguments for Remove-NM

1: **function** RemoveNotMinority(G, H, L, f)
2: **for all** $x \in V(G)$ and $a \in L(x)$ **do**
3: Set $flag[x, a] = true$
4: **for all** $x, y \in V(G), (a, b) \in L(x, y)$ **do**
5: $flag[x, a](y, b) = true$
6: **for all** $z \in V(G) \setminus \{x, y\}, c \in L(z)$ with $(a, c) \in L(x, z), (b, c) \in L(y, z)$ **do**
7: set $flag[x, y, a, b](z, c) = true$
▷ flag refers to the $flag[x, a]$ and also refers to $flag[x, y, a, b]$
8: In any other case $flag$ is set to false, i.e. $flag[x, a](y, b) = false$ when $(a, b) \notin L(x, y)$.
9: Initiate an empty vector V ▷ is used for decomposing the input into bi-cliques
10: Remove-NM$(G, L, f, flag, V)$

The RemoveNotMinority takes G, H, L, f as input. It defines flag variables that are used to eliminate some of the wrong choices. We look at them as hybrid variable. $flag[x, a]$ being false means that there is no L-homomorphism from G to H that maps x to a. This is because we have made a test and the answer of the test was false (no solution). $flag[x, y, a, b]$ defines a set of boolean variables, with the intention that we are looking for an L-homomorphism from G to H that maps x to a and y to b. If $flag[x, a](y, b)$ is false then it means there is no such a homomorphism.
Having $flag[x, y, a, b](z, c) = false$ means that there is no L-homomorphism that maps x to a, y to b and z to c. Initially all these variable are true. The vector V is used for decomposing the instance into bi-cliques.

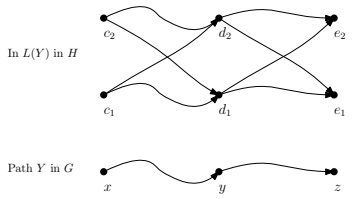

Figure 1: An example of a Bi-clique

Algorithm 4 updating f so that it remains a homomorphism of $G \times H^k$ to H consistent with L and for every $x \in V(G)$, $a', b' \in L(x)$, we have $f(x; a'^k, b') = b'$

function REMOVE-NM($G, L, f, flag, V$)
  Input: digraphs G, H, lists L and, weak-nu homomorphism $f : G \times H^k \to H$
  If $|L(x)| = 1$ then assign $L(x)$ to x and PREPROCESSING(G, L)
  If $G \times_L H$ is not connected then consider each valid connected component separately
  if there is no non-minority pair **then**
    $g =$ REMOVEMINORITY(G, H, L, f)
    return g
  while exists a non-minority pair **do**
    if exists a cut vertex (x, a) **then**
      $(L, f, flag) =$ CUT-VERTEX-CASE($G, L, f, x, a, flag$)
    else
      $(L, f, flag) =$SYMMETRIC-DIFFERENCE($G, L, f, flag$)
      $(L, f, flag) =$BI-CLIQUE-INSTANCES($G, L, f, flag, V$)
  $(L, f, flag) =$ MAKING-F-CONSISTENT($G, L, f, flag$)
  for all $x \in V(G)$, $a \in L(x)$ **do**
    if $\exists y \in V(G)$ s.t. $y \neq x$ & $\forall (a, d_i) \in L(x, y)$, $flag[x, a](y, d_i) = false$ **then**
      $flag[x, a] = false$
  for all $x \in V(G), a \in L(x)$ **do**
    if a is not relevant OR $flag[x, a] = false$ **then** Remove a from $L(x)$.
  PREPROCESSING(G, L)
  return $flag$

The REMOVE-NM takes $G, L, f, flag, V$ and it checks whether there are lists with only one element and in this case the decision is clear. It also handle each connected component of $G \times_L H$ independently. If there is no non-minority pair then it calls REMOVEMINORITY. Otherwise it starts with the cut vertices in $G \times_L H$ first. It is clear that if the test for cut vertex (x, a), fails then the remaining instance has different components and each connected component is handled separately.

If the test $T_{x,c}$ for a smaller instance succeeds then it means there exists an L homomorphism g from G' (a sub-digraph of G) to H. Here G' is the digraph induced by the first coordinates on the vertices $G_L(x, a, c)$. We modify f on the vertices G' and lists L using g in function UPDATE-F and at the end there is no vertex $w \in G \times H^k$ with $f(w) = a$.

If there is no cut vertex then REMOVE-NM calls the SYMMETRIC-DIFFERENCE function. In this function we make a smaller test which actually tests a necessary condition for having a homomorphism that maps x to c and y to d_1 and z to e_1 in a sub-digraph of G. For every $v \in V(G)$, the list $L'(v)$ is a subset $L(v)$ consisting of vertices i such that $(i, d_2) \notin L(v, y)$.

After that we handle the Bi-Clique instances. At the end of while loop at line (8) we need to make f consistent. This is we are going to remove some vertex a from $L(x)$ because the test $T_{x,a}$ fails. Now if there exists $a_1, a_2 \in L(x)$ such that $f(x; a_1^k, a_2) = a$ then f will no longer be consistent with the lists. Therefore we argue that at least one of the T_{x,a_1}, T_{x,a_2} should be false. In the MAKING-F-CONSISTENT we check which of the a_1, a_2 should also be removed from $L(x)$. At line 15-16 we make a decision for $flag[x, a]$ if there exists a vertex y such that there is no homomorphism that maps x to a and y to any of the elements in $L(y)$.

1: **function** CUT-VERTEX-CASE($G, L, f, x, a, flag$)
2: Let (L', f')= SMALL-INSTANCE$(L, f, x, a, flag)$
3: $flag$ = REMOVE-NM$(G, L', f', flag, \emptyset)$
4: g = REMOVEMINORITY(G, H, L', f')
5: **if** g is not empty **then**
6: UPDATE-F$(g, L, \emptyset, \emptyset, x, a, f)$
7: **else** $flag[x, a] = false$ and remove (x, a) from $G \times_L H$.
8: **return** $L, f, flag$

Here we create an instance in which $L'(x) = a$ and $L'(y)$ contains element e' with $(a, e) \in L(x, y)$. We also take into account that it is conceivable that y can be mapped to e as otherwise $flag[y, e]$ has been set to be false at some earlier steps of the algorithm.

```
1: function SMALL-INSTANCE(G, L, f, x, a, flag )
2:     Create new empty lists L'.
3:     for all y ∈ V(G) do
4:         L'(y) = { e | (a, e) ∈ L(x, y) & e is relevant & flag[y, e] }.
5:     Set f' = f|_{L'}                                   ▷ see definitions 3.4
6:     return (G, L', f')
```

```
1: function UPDATE-F(g, L, x_0, c_0, x, c, f )
2:     if x_0 = ∅ then set L' = L
3:     else
4:         Set L'(x_0) = c_0
5:         for all y ∈ V(G) do L'(y) = {e|(c_0, e) ∈ L(x_0, y)}
6:     for all a ∈ L(x) s.t. a ≠ c, (x, a), (x, c) are in the same component of G ×_{L'} H do
7:         Construct G_{L'}(x, a, c).
8:         for all (y, d, g(y)) ∈ G_{L'}(x, a, c) do
9:             for all w_1 = (y; a_1, a_2, ..., a_k) with f(w_1) = d do set f(w_1) = g(y)
```

1: **function** SYMMETRIC-DIFFERENCE(G, L, f)
2: Create new $flag$ variables and set them to be true.
3: Create new empty lists L'.
▷ Considering all the forks
4: **for all** $x, y \in V(G)$, $c_1 \in L(x)$, $d_1, d_2 \in L(y)$ with
5: $flag[x, c_1](y, d_1)$ & $(c_1, d_1), (c_1, d_2) \in L(x, y)$ **do**
6: Set $L'(x) = c$, $L'(y) = d_1$
7: **for all** $v \in V(G)$ s.t. for every $i \in L(v)$, $(d_2, i) \notin L(y, v)$ **do**
8: $L'(v) = \{i | (c_1, i) \in L(x, v), (d_1, i) \in L(y, v), \& flag[x, c_1](v, i), flag[y, d_1](v, i), flag[v, i]\}$
9: Let G' be the induced sub-digraph of G with vertices v s.t. $L'(v) \neq \emptyset$
10: $flag =$ REMOVE-NM$(G', L', f', flag, \emptyset)$
11: $g =$ REMOVEMINORITY(G', H, L', f')
12: **if** g is empty **then** $flag[x, c_1](y, d_1) = flag[y, d_1](x, c_1) = false$.

▷ Definition : $x, y, z \in V(G)$, $c_1 \in L(x)$, $d_1, d_2 \in L(y)$, $e_1 \in L(z)$ make a diamond **when** $(c_1, d_1), (c_1, d_2) \in L(x, y)$ & $(c_1, e_1) \in L(x, z)$ & $(d_1, e_1), (d_2, e_1) \in L(y, z)$
▷ use the same flag variables
Note that if $flag[x, c_1](y, d_1) = false$ then $flag[x, y, c_1, d_1](z, e_1) = false$
13: Create new empty lists L'.
14: **for all** $x, y, z \in V(G)$, $c_1 \in L(x)$, $d_1, d_2 \in L(y)$, $e_1 \in L(z)$ with
15: $flag[x, y, c_1, d_1](z, e_1)$ & $x, y, z, c_1, d_1, d_2, e_1$ make a *diamond* **do**
16: Set $L'(x) = c$, $L'(y) = d_1$, and $L'(z) = e_1$.
17: **for all** $v \in V(G)$ s.t. for every $j \in L(v)$, $(d_2, j) \notin L(y, v)$ **do**
18: $X = \{i | flag[x, c_1](v, i), flag[y, d_1](v, i), flag[z, e_1](v, i), flag[v, i]$ all true $\}$
19: $L'(v) = \{i | (c_1, i) \in L(x, v), (d_1, i) \in L(y, v), (e_1, i) \in L(z, v),$ and $i \in X\ \}$
20: Let G' be an induced sub-digraph of G on the vertices v such that $L'(v) \neq \emptyset$
21: $flag =$ REMOVE-NM$(G', L', f', flag, \emptyset)$
22: $g =$ REMOVEMINORITY(G', H, L', f')
23: **if** g is empty **then** $flag[x, y, c_1, d_1](z, e_1) = false$.
▷ Cleaning up the flag variables
24: **for all** $x, y \in V(G)$, $a_1 \in L(x), b_1 \in L(y)$ **do**
25: **if** $\exists z \in V(G)$ s.t.
$\forall (a_1, d_i) \in L(x, z), (b_1, d_i) \in L(y, z)$, $flag[y, b_1](z, d_i) = false$ **then**
26: $flag[x, a_1](y, b_1) = false$
27: **return** $flag$

1: **function** BI-CLIQUE-INSTANCES($G, L, f, flag, V$)
2: $(x, a, c, y, d, d', z, e_1, e_2)$ = BI-CLIQUE-1$(G, L, f, flag, V)$
▷ Bi-Clique-1 is call by reference for vector V, it updates the current vector V
3: Let $S_1 = \widehat{f}_{x,\{a,c\}}$, $S_2 = \widehat{f}_{y,\{d,d'\}}$, $S_3 = \widehat{f}_{z,\{e_1,e_2\}}$
4: **for all** $c_0, c_1 \in S_1$ s.t. $flag[x, c_1]$ & $f(x; c_0'^k, c_0) = c_1 \neq c_0$ for some $c_0' \in S_1$ **do**
5: **for all** $d_1, d_2 \in S_2$ s.t. $flag[x, c_1](y, d_2)$ **do**
6: **for all** $e_1 \in S_3$ **do**
7: $test_1 = flag[x, y, c_1, d_1](z, e_1)$
8: $test_2 = flag[x, y, c_1, d_2](z, e_1)$
9: **if** $test_1$ and $test_2$ **then**
▷ it means we can replace $d_1, d_2, \ldots, d_i$ with one element
10: (L', f', d) = RESTRICTED-INSTANCE$(G, L, f, x, c_1, y, d_1, d_2, z, e_1, flag)$
11: $flag[x, y, c_1, d]$ = REMOVE-NM$(G', L', f', flag, V)$
12: g = REMOVEMINORITY(G', H, L', f')
13: **if** g is not empty **then**
14: UPDATE-F(g, L, x, c_1, y, d, f) and update V accordingly
15: **else**
16: **for all** $d_i \in \widehat{f}_{y,\{d_1,d_2\}}$ **do** $flag[x, y, c_1, d_i](z, e_1) = false$
17: Remove the last element of V
▷ no more such z, e_1 and hence each component is treated separately
18: (G', L', f') = SMALL-INSTANCE-2$(G, L, f, x, c_1, y, d_1, flag$)
19: $flag[x, y, c_1, d_1]$ = REMOVE-NM$(G', L', f', flag, V)$
20: g = REMOVEMINORITY(G', H, L', f')
21: **if** g is not empty **then**
22: UPDATE-F$(g, L, x, c_1, y, d_1, f)$
23: **else**
24: $flag[x, c_1](y, d_1) = flag[y, d_1](x, c_1) = false$
25: **return** $(L, f, flag)$

Let $x, y \in V(G)$ and $c_1, c_2 \in L(x)$ and $d_1, d_2 \in L(y)$ induce a Bi-Clique in $G \times_L H$. Let $e_1 \in L(z)$ such that $(c_1, e_1) \in L(x, z)$, $(c_1, d_1), (c_1, d_2) \in L(x, y)$ and $(d_1, e_1), (d_2, e_1) \in L(z)$. Then in the Bi-Clique-Instance we want to see whether it is possible to replace $d_1, d_2 \in L(z)$ by some other elements, say for example $f(y; d_1^k, d_2)$. To do that we have already made a smaller test, say $test_1$ in which we checked whether $flag[x, y, c_1, d_1](z, e_1)$ is true or false. We consider the list $L'(v)$ consists of the vertices that are reachable from $d_1 \in L(y)$, $c_1 \in L(x)$, $e_1 \in L(z)$ but not reachable from $d_2 \in L(y)$. This mean we check whether there exists an L'-homomorphism from G' (an induced sub-digraph of G) to H that maps x to c_1 and z to e_1 (if such e_1 exists) and y to d_1. Analogously we have made a smaller test, say $test_2$ in which we checked whether $flag[x, y, c_1, d_2](z, e_1)$ is true or false. If both $test_1, test_2$ are true then we construct an instance (Restricted-Instace) in which we replace $\widehat{f}_{y,\{d_1,d_2\}}$ by just one element in $\widehat{f}_{y,\{d_1,d_2\}}$ and continue. Note that if there is no such e_1 then it means instances $I_1(x, c_1, y, d_1)$ (assigning x to c_1 and y to d_1) and $I_2(x, c_1, y, d_2)$ create different connected components in $G \times_L H$ and each of them is treated separately. To do so, we call small-instance-2 function that creates lists L' for which $L'(x) = c_1$ and $L'(y) = d_1$ and ask whether such a homomorphism exists. If the answer is yes then we update f by calling update-f function. Otherwise we set $flag[x, c_1](y, d_1) = false$.

1: **function** Restricted-Instance$(G, L, f, x, c_1, y, d_1, d_2, z, e_1, flag\ )$
2: Create new empty lists L'
3: Let d be the unique vertex in $L(y)$ which is at the end of $\widehat{f}_{y,\{d_1,d_2\}}$. This means $d \in \widehat{f}_{y,\{d_1,d_2\}}$ and for every $d' \in f(y; d'^k, d) = d$ or if no such d exists then there are $d_i, d_{i+1}, \ldots, d_t$ s.t. $f(y; d_j^k, d_{j+1}) = d_{j+2}$, $i \leq j \leq t$ where $d_{t+1} = d_i$, $d_{t+2} = d_{i+1}$.
4: $L'(x) = \{c\}$, $L'(y) = \{d\}$
5: **for all** $v \in V(G) \setminus \{x, y, z\}$ **do**
6: $X = \{i | flag[x, c_1](v, i), flag[y, d_1](v, i), flag[z, e_1](v, i), flag[v, i]$ all true $\}$
7: $L'(v) = \{i |\ (c_1, i) \in L(x, v), (d, i) \in L(y, v), (e_1, i) \in L(z, v)\ \&\ i \in X\ \}$
8: Set $f' = f|_{L'}$ ▷ see definitions 3.4
9: **return** (L', f', d)

function BI-CLIQUE-1$(G, L, f, flag, V\ )$
  if V is empty **then**
    Let x, y, z be three distinct vertices of G s.t. the following hold :
      $\exists a, b \in L(x)$ s.t. a is relevant & $f(x; b^k, a) = c \neq a$
      $d, d' \in L(y)$ where x, a, c together with y, d, d' induce a minimal bi-clique in L (here d' could be d)
      $flag[x, c] = true$, $flag[x, c](y, d) = true$
      Let e_1, e_2 be two vertices in $L(z)$ s.t. y, d, d', z, e_1, e_2 induce a minimal bi-clique and $(a, e_1), (a, e_2) \in L(x, z)$. If there is a choice then $(c, e_1), (c, e_2) \in L(x, z)$.
        ▷ Note that such y, d must exist otherwise different connected component
      Add $(x, a, c, y, d, d', z, e_1, e_2)$ to the end of V
  else
    Let $(x, a, c), (y, d, d')$ be the last two elements of V
    Let $z \neq x, y$ be a vertex of G s.t the following hold :
      $\exists e_1, e_2 \in L(z)$ where y, d, d' together with z, e_1, e_2 induce a minimal bi-clique in L and $(a, e_1), (a, e_2) \in L(x, z)$ and $flag[x, y, c_1, d_1](z, e_1) = true$. If there is a choice then $(c, e_1), (c, e_2) \in L(x, z)$ (here e_1 could be e_2)
        ▷ if no such z, e_1, e_2 exist then different component
    Add (z, e_1, e_2) to the end of V

---

function SMALL-INSTANCE-2$(G, L, f, x, a, y, d, flag\ )$
  Create new empty lists L'.
  for all $v \in V(G) \setminus \{x, y\}$ **do**
    $X = \{i | flag[x, a](v, i), flag[y, d](v, i), flag[v, i]$ are all true $\}$
    $L'(v) = \{\ i \mid (a, i) \in L(x, v),\ (d, i) \in L(y, v)$ & i is relevant $\}$.
  Set $f' = f|_{L'}$ ▷ see definitions 3.4
  return (G, L', f')

---

function MAKE-F-CONSISTENT$(G, L, f, flag\ )$
  while $\exists\ a, b, c \in L(x)$ s.t. $f(x; b^k, a) = c$ & $flag[x, a], flag[x, b]$, $\neg flag[x, c]$ **do**
    Let (L', f')= SMALL-INSTANCE$(L, f, x, a, flag)$
    $flag$ = REMOVE-NM$(G, L', f', \emptyset)$
    g = REMOVEMINORITY(G, H, L', f')
    if g is not empty **then**
      for all $d \in L(x)$ s.t $f(x; d^k, a) = c$ **do** $flag[x, d] = false$
    else $flag[x, a] = false$
  return $L, f, flag$

3.2 Minority Algorithm (RemoveMinority)

In this section we show that once the minority case has been reached in our main algorithm, we can reduce to an already solved setting for homomorphism testing – namely that of the Mal'tsev case. We note that this section is independent of the rest of the algorithm.

Note that at this point for every $a, b \in L(x)$ we have $f(x; b^k, a) = a$ and in particular when $a = b$ we have $f(x; a, a, \ldots, a) = a$ (idempotent property). This is because when a is in $L(x)$ then it means the Remove-NM procedure did not consider a and in fact did not change the value of $f(x; ...)$ from a to something else. Note that for the argument below we just need the idempotent property for those vertices that are in $L(x)$, $x \in V(G)$.

A ternary polymorphism h' is called Maltsev if for all $a \neq b$, $h'(a, b, b) = h'(b, b, a) = a$. Note that the value of $h'(b, a, b)$ is unspecified by this definition.

Let G and H be as input to Algorithm 1, and suppose line 6 of the algorithm has been reached. We define a homomorphism $h : G \times H^3 \to H$ consistent with the lists L by setting $h(x; a, b, c) = f(x; a, b, b, \ldots, b, c)$ for $a, b, c \in L(x)$. Note that because f has the minority property for all $x \in G$, $a, b \in L(x)$, h is a Maltsev homomorphism consistent with the lists L.

Note that for the argument below we just need the idempotent property for those vertices that are in $L(x)$, $x \in V(G)$.

Let G' be the structure obtained from G by making each arc a different binary relation. In other words, G' has vertices $V(G)$ and $|E(G)|$ binary relations $R_e, e \in E(G)$, where $R_e = \{xy\}$ if e is the arc $e = xy$.

Let H' be the structure where $V(H')$ is the disjoint union of $L(x), x \in V(G)$, and there are also $|E(G)|$ binary relations $S_e, e \in E(G)$, where S_e is the set of all ordered pairs ab with $ab \in E(H), a \in L(x), b \in L(y)$, where $e = xy$. Note that $|V(H')| \geq |V(G')|$ if each $L(x)$ is non-empty. This may seem unusual for the homomorphism setting, but is certainly allowed.

Now note that there is an L-homomorphism of G to H (i.e., a list homomorphism consistent with lists L) if and only if there is a homomorphism of G' to H'. Homomorphisms of such structures are mappings $f : V(G') \to V(H')$ such that $xy \in R_e$ implies $f(x)f(y) \in S_e$ for all $e \in E(G)$.

Finally, note that the structure H' has a Maltsev polymorphism h' of the ordinary kind. Indeed, let h_x be our Maltsev polymorphisms defined on $L(x)$ by setting $h_x(a, b, c) = h(x; a, b, c)$. We let $h'(a, b, c) = h(x; a, b, c)$ if a, b, c are from the same $L(x)$, and for a, b, c not from the same $L(x)$ define $h'(a, b, c) = a$ unless $a = b$, in which case define it as $h'(a, b, c) = c$. The definition ensures that h is Maltsev. To check it is a polymorphism, note that $aa', bb', cc' \in S_e$ is only possible if $a, b, c \in L(x), a', b', c' \in L(y)$, where $e = xy$. For those, we have the polymorphism property by assumption.

Now we have a structure with a Maltsev polymorphism, so the Bulatov-Dalmau [BD06] algorithm applies and solves the homomorphism problem. Note that Corollary 4.2 of the Bulatov-Dalmau paper explicitly mentions that it is polynomial in both the sizes of G and

H.

Therefore we have the following theorem.

Theorem 3.10 *Suppose $h : G \times H^k \to H$ is a minority homomorphism consistent with lists L on G. Then the existence of an L-homomorphism of G to H can be decided in polynomial time.*

Remark : We have communicated with the authors of [BD06] and they confirmed that indeed we can apply their algorithm as explained above. We note that it is also possible to give a direct algorithm for the minority case that is similar to how we handle the "not minority" case.

4 Proofs

4.1 PreProcessing and List Update

We first show that the standard properties of consistency checking remain true in our setting – namely, that if the PreProcessing algorithms succeed then f remains a homomorphism consistent with the lists L if it was before the PreProcessing.

Lemma 4.1 *If f is a homomorphism of $G \times H^k \to H$ consistent with L then f is a homomorphism consistent with L after running the pre-processing.*

Proof: We need to show that if $a_1, a_2, \ldots, a_k$ are in $L(y)$ after the pre-processing then $f(y; a_1, a_2, \ldots, a_k) \in L(y)$ after the pre-processing. By definition vertex a is in $L(y)$ after the pre-processing because for every oriented path Y (of some length m) in G from y to a fixed vertex $z \in V(G)$ there is a vertex $a' \in L(z)$ and there exists a walk B in H from a to a' and congruent with Y that lies in $L(Y)$.

Let $a'_1, a'_2, a'_3, \ldots, a'_k \in L(z)$. Let A_i, $1 \leq i \leq k$ be a walk from a_i to a'_i in $L(Y)$ and congruent to Y. Let $A_i = a_i, a^i_1, a^2_i, \ldots, a^m_i, a'_i$ and let $Y = y, y_1, y_2, \ldots, y_m, z$.

Since f is a homomorphism consistent with L before the pre-processing, $f(y; a_1, a_2, \ldots, a_k)$, $f(y_1; a^1_1, a^1_2, \ldots, a^1_k), \ldots,$ $f(y_i; a^i_1, a^i_2, \ldots, a^i_k), \ldots, f(y_m; a^m_1, a^m_2, \ldots, a^m_k), f(z; a'_1, a'_2, \ldots, a'_k)$ is a walk congruent with Y. This would imply that there is a walk from $f(y; a_1, a_2, \ldots, a_k)$ to $f(z; a'_1, a'_2, \ldots, a'_k)$ congruent with Y in $L(Y)$ and hence $f(y; a_1, a_2, \ldots, a_k) \in L(y)$. $\diamond$

By a similar argument as in the proof of Lemma 4.1 we have the following lemma.

Lemma 4.2 *If f is a homomorphism of $G \times H^k \to H$, consistent with L and $a_1, a_2, \ldots, a_k \in L(x)$, $b_1, b_2, \ldots, b_k \in L(y)$, and $(a_i, b_i) \in L(x, y)$, $1 \leq i \leq k$, after pre-processing then $(f(x; a_1, a_2, \ldots, a_k), f(y; b_1, b_2, \ldots, b_k)) \in L(x, y)$ after the pre-processing.*

4.2 RemoveNotMinority Correctness Proof

The main argument is proving that after RemoveNotMinority algorithm, f still is a homomorphism consistent with the lists and has weak-nu property (Lemma 4.3). Moreover, after RemoveNotMinority there still exists a homomorphism from G to H if there was one before RemoveNotMinority.

Lemma 4.3 *If f is a homomorphism of $G \times H^k \to H$, consistent with L and with weak-nu property before RemoveNotMinority then the modified f remains a homomorphism consistent with L and with weak-nu property afterwards. Moreover, if there is a homomorphism $\psi : G \to H$ then there is a homomorphism from G to H after RemoveNotMinority.*

Proof: It is enough to show the following :
f is a homomorphism of $G \times H^k \to H$ consistent with the lists (with weak-nu property) after removing a from $L(x)$ in the while loop in Algorithm 4 (Remove-NM).

We need to address items 1,2,3,4 below.

1. The weak-nu property is preserved.
2. The adjacency property is preserved: for an arbitrary arc $yz \in A(G)$ ($zy \in A(G)$) and for every $a'_1, a'_2, \ldots, a'_k \in L(y)$ and $b'_1, b'_2, \ldots, b'_k \in L(z)$ where $a'_i b'_i \in A(H)$ ($b'_i a'_i \in A(H)$), $1 \leq i \leq k$, we have $f(y; a'_1, a'_2, \ldots, a'_k) f(z; b'_1, b'_2, \ldots, b'_k) \in A(H)$ ($f(z; b'_1, b'_2, \ldots, b'_k) f(y; a'_1, a'_2, \ldots, a'_k) \in A(H)$).
3. There exists a homomorphism from G to H after Remove-NM if there exists one before.
4. The Running time of Remove-NM function is polynomial .

Proof of 1 : Since in the Update-f function we change the value $f(y; a'_1, a'_2, ..., a'_k)$ from a_2 to $f(y; a_1^k, a_2)$ for every k-tuple $a'_1, a'_2, \ldots, a'_k \in L(y)$, we change $f(y; b_1, b_1, \ldots, b_1, b_2) = f(y; b_1, b_1, \ldots, b_2, b_1) = \cdots = f(y; b_2, b_1, \ldots, b_1)$ to the same value. Therefore f still has the weak k-nu property.

Proof of 2 : Let $a_{k+1} = f(w_1 = (y; a'_1, a'_2, \ldots, a'_k))$ and $b_{k+1} = f(w_2 = (z; b'_1, b'_2, \ldots, b'_k))$. We need to show that $a_{k+1} b_{k+1} \in A(H)$ when $w_1 w_2$ is an arc of $G \times H^k$. Note that a_{k+1} is the last value of $f(y; a'_1, a'_2, \ldots, a'_k)$ and b_{k+1} is the last value of $f(x; b'_1, b'_2, \ldots, b'_k)$. Suppose a_1 was the initial value of $f(w_1)$ and b_1 is the initial value of $f(w_2)$. Note that $yz \in A(G)$ and $a_1 b_1 \in A(H)$ because f is initially a homomorphism consistent with the lists L.

Observe that the value of $f(y; a'_1, a'_2, \ldots, a'_k)$ could have been changed several times before finally being set to a_{k+1}. We show that at each step of the Algorithm 4 if $f(w_1)$ changes from a_1 to some new value a_2 then $f(w_2)$ is also changes from b_1 to some out-neighbor of a_2 say b_2.

We also remember that at each step of the recursive call we deal with some vertex x of G and two vertices $a, b \in L(x)$ such that $f(x; b^k, a) = c \neq a$, and $f(x; c, c, \ldots, c) = c$ considered in Algorithm 4 (line 4). Moreover, the assumption is that f is a homomorphism consistent with the lists L and if $f(y; e'_1, e'_2, \ldots, e'_k) = e_1$ then $f(y; e_1, e_1, \ldots, e_1) = e_1$

We need to show that after each step if there exists an L-homomorphism g obtained from Remove-NM function, then f is still a homomorphism consistent with the lists.

Suppose in function Update-F line 4, we change $f(w_1)$ from d to $g(y)$. Let Y be an oriented path from x to y on the digraph G' obtained from the first coordiates of $G_L(x, a, c)$. Therefore there exists a walk A_1 from a to d in $L(Y)$ and congruent with Y. Now $A'_1 = A_1 b_1$ is a walk in $L(Yz)$ and congruent with Yz (obtained by adding arc yz to the end of Y).

Because of PreProcessing, there exists some vertex $c_1 \in L(y)$ such that $(c, c_1) \in L(x, y)$ and hence there exists a walk C_1 in $L(Y)$ from c to c_1 and congruent to A_1. Now since yz is an arc, c_1 must have some neighbor d_1 in $L(z)$ and hence $L'(z)$ is not empty. First assume that $g(y)b_1 \notin A(H)$, and hence by definition of $G_L(x, a, c)$ we have $(z, b_1, c_1) \in G_L(x, a, c)$. This would mean we change $f(w_2)$ from b_1 to $g(z)$ and since $g(y)g(z)$ is an arc, we have $f(w_1)f(w_2) \in A(H)$. If $g(y)b_1$ is an arc then it means we did not change $f(w_2)$ and hence again $f(w_1)f(w_2) = g(y)b_1 \in A(H)$.

In what follows we show the correctness of Make-F-consistence function. If we have set $flag[x, c] = false$, after running Small-Instance then at least one of the $flag[x, a]$, $flag[x, b]$, where $c = f(x; b^k, a)$ should be false after running Small-Instance. This is equivalent to show the following.

Claim 4.4 *If a, b remain in $L(x)$ after step 5 of function* Make-f-consistence *then $c = f(x; b^k, a) \in L(x)$.*

Proof: We need to show that if $flag[x, a] = flag[x, b] = true$ after running line 5 of function Make-f-consistence then $flag[x, c]$ is also true and hence c will be in $L(x)$.

Let (L'_a, f'_a) be an instance obtained from Small-Instance $(G, L, f, x, a, flag)$ and (L'_b, f'_b) be an instance obtained from Small-Instance $(G, L, f, x, b, flag)$.

Suppose g_a, an L'_a homomorphism from G to H exists and g_b, an L'_b homomorphism from G to H also exists. Suppose $L'_a(y)$ has some vertex a_1. This means there exists an oriented path Y in G and a walk A_1 in $L(Y)$ from $a \in L(x)$ to a_1. Since we have performed PreProcessing and $b \in L(x)$, there must exist a vertex $b_1 \in L(y)$ and a walk B_1 in $L(Y)$ from b to b_1 which is congruent to A_1.

This means if $L'_a(y)$ is not empty then $L'_b(y)$ is not empty. Now there exists a directed path W in $G \times H^k$, from $w = (x; b^k, a)$ to $(y; b_1^k, a_1)$ with n vertices, such that the i^{th} vertex of W is of form $w_i = (v_i; d_i^k, e_i)$ where v_i is the i^{th} vertex of Y and d_i, e_i are the i^{th} vertices of B_1, A_1 respectively. Now since f is a homomorphism, $f(w_1), f(w_2), \ldots, f(w_n)$ is a walk in $L(Y)$, from c to $c' \in L(z)$, and congruent with Y. This would imply that $(c, c') \in L(x, z)$ and hence $L'(y)$ is not empty. Here L' is the list of vertices that are constructed in test $T_{x,c}$. Conversely, if there is a $c' \in L'(y)$ then because of the PreProcessing $L'_a(y)$ and $L'_b(y)$ are not empty. Now define mapping g from G to H with $g(y) = f(y; g_b(y)^k, g_a(y))$. Note

that $g(x) = c$. Since f is a homomorphism from G to H^k and consistent with the list L, $g(y) \in L'(y)$. Let $yz \in A(G)$ be an arbitrary arc. Then $(y; g_b(y)^k, g_a(y))(z; g_b(z)^k, g_a(z))$ is an arc of $G \times H^k$ and hence $g(y)g(z) \in A(H)$. This implies that g is an L' homomorphism from G to H that maps x to c, a contradiction. $\diamond$

Proof of 3 : The algorithm 4 make a test $T_{x,c}$ on a smaller instance of the problem to decide whether f can be modified in such a way that $f(x; \dots)$ is set to c from a, i.e. a is not a relevant vertex in $L(x)$. We show the following.

Claim 4.5 *Let G', L', f' be an instance of the problem to perform the test $T_{x,c}$. Then the following hold.*

I. Suppose $T_{x,c}$ passes. If there exists a L-homomorphism from G to H that maps x to a where $(x, a), (x, c)$ are in the same component $G \times_L H$ then there is one that maps x to c.

II. If $T_{x,c}$ fails then there is no L-homomorphism from G to H that maps x to c.

Proof: Proof of I : It is enough to show that if there is a homomorphism $\psi : G \to H$ with $\alpha(x) = a$ then there is a homomorphism from G to H after removing a from $L(x)$ in Algorithm 4. Since the test $T_{x,c}$ has been passed, there exists an L'-homomorphism g from G' to H that is obtained from solving instance $I_1 = (L', f')$. Since $(x, a), (x, c)$ are in the same connected component of $G \times_L H$ and α is a homomorphism from G' to H, for every vertex $v \in V(G)$, $L'(v) \neq \emptyset$. This would imply that $G' = G$ and hence we have a L-homomorphism α from G to H. $\diamond$

Proof of II : We use induction on the size of the instance, $|G||L|$, where $|L| = \sum_{v \in (G)} |L(v)|$. We first look at the instance $I_1 = (L', f')$ considered in Small-Instance$(G, L, f, x, a, flag)$. The argument for other functions is similar because in each recursive call we may encounter all the functions inside the Algorithm 4.

First suppose in constructing the L' lists (see function Small-Instance (line 3)) every vertex $e \in L(y)$, e is a relevant vertex in $L(v)$ and $flag[y, e]$ is true. Therefore $L'(y) = \{e|(e, a) \in L(x, y)\}$ and hence none of the smaller test has actually affected any of the flag variables. In this scenario the only problem would be that the RemoveMinority returns an empty homomorphism and hence there is no homomorphism from G to H that maps x to a.

Now consider the case in which for some vertex $y \in V(G)$, $e \in L(y)$ where $(a, e) \in L(x, y)$, $L'(y)$ does not contain e because e is not a relevant vertex. Note that in this case we may assume that there exists some homomorphism ψ that maps x to a and y to e. Now the reason that the test $T_{x,a}$ fails is because we did not include e into the $L'(y)$. The reason that e is not relevant is because at some earlier step of algorithm some test $T_{y,e}$ has been passed (or some test $T_{z,e'}$ has been passed and (y, e) and (z, e') where in the same connected component of $G \times_L H$) and hence according to function Update-F we have modified f so that no

$w = (y; \dots) \in G \times H^k$, $f(w) = e$. For simplicity we may assume test $T_{y,e}$ has been passed. Now according to (I) if there exists a homomorphism σ from G to H that maps y to e then there must be a homomorphism σ' from G to H that maps y to some other vertex $e' \in L'(y)$. We may assume that $\sigma'(x) = a$ and $\sigma'(y) = e'$. This would imply that $(a, e') \in L(x, y)$. Therefore in the instance I_1 we may assume that $e \notin L'(y)$. Therefore I_1 has a smaller size and by induction hypothesis if the test $T_{x,a}$ fails then there is no homomorphism form G to H that maps x to a.

We continue by considering another reason. Suppose in instance I_1, $L'(y)$ does not include e where $(a, e) \in L(x, y)$ because $flag[y, e] = false$ and this may have caused the test $T_{x,a}$ to fail. The reason that $flag[y, e] = false$ is because of induction hypothesis when we call Small-Instance in the earlier stage of the procedure or because of the other parts of the algorithm which we address below.

Handling Bi-Cliques : The heart of the algorithm is the following: We start with some vertex x of G and some $c \in L(x)$. We look at the instance of the problem $I_1 = (G', L', f')$ obtained where in $L'(x) = \{c\}$ and for every vertex $v \in V(G)$, $L'(v) = \{d | (c, d) \in L(x, v)\}$. Now we look at some vertex $y \in V(G')$ according to the construction of $G_L(x, a, c)$. The Algorithm looks at pairs $c_1, c_2 \in L'(y)$ and it makes a decision to see whether f' can be modified so that $f'(w_1 = (y, e_1, e_2, \dots, e_k)) \neq c_1$ and instead the $f'(w_1)$ is set to $f'(y; c_2^k, c_1)$ if it possible (by the argument in Claim 4.6 (1), $(c, f(y; c_2^k, c_1)) \in L(x, y)$). If this modification is not possible then it means if there exists a homomorphism $\psi : V(G) \to V(H)$ that maps x to c then ψ does not map y to $c' = f'(y; c_2^k, c_1)$ and hence $flag[x, c](y, c')$ is set to false. The Algorithm tries different pairs and there must be at least one pair for which this transformation is possible or at the end it reaches to a point that for each $c_1, c_2 \in L(y)$, $f'(y; c_2^k, c_1) = c_1$ and $f'(y; c_1^k, c_2) = c_2$.

Claim 4.6 *Let $(a, d), (c, d), (a, d'), (c, d') \in L(x, y)$. Here d could be the same as d' but $a \neq c$. Let $c_1 \in \widehat{f}_{x,\{a,c\}}$ and $d_1 \in \widehat{f}_{y,\{d,d'\}}$ be an arbitrary elements. Let $e_1, e_2 \in L(z)$ such that $(a, e_1), (a, e_2), (c, e_1), (c, e_2) \in L(x, z)$ and $(d, e_1), (d, e_2) \in L(y, z)$. Then the following hold :*

(a) $(c_1, d_1) \in L(x, y)$.

(b) $(c_1, e_1), (c_1, e_2) \in L(x, z)$ *and* $(d_1, e_1), (d_1, e_2) \in L(x, y)$,

(c) $(c_1, e_3) \in L(x, z)$ *and* $(d_1, e_3) \in L(y, z)$ *where* $e_3 = f(z; e_1^k, e_2)$.

Proof: Proof of (a) : Let $(a_1, b_1) \in L(x, y)$ and $(a_2, b_2) \in L(x, y)$.

Let Y be an arbitrary oriented path from x to y in G and let A_1 be a walk in $L(Y)$ from a_1 to b_1 and A_2 be a walk in $L(Y)$ from a_2 to b_2 such that A_1, A_2 are congruent.

Let $Y = y_1, y_2, \dots, y_m$ where $y_1 = x$ and $y_m = y$. Let $a_3 = f(x; a_1^k, a_2)$ and $b_3 = f(y; b_1^k, b_2)$. Now consider the walk $A_3 : f(y_1; r_1^k, s_1), \dots, f(y_i; r_i^k, s_i), \dots, f(y_m; r_m^k, s_m)$, where r_i is the

i^{th} vertex of A_1 and s_i is the i^{th} vertex of A_2 and y_i is the i^{th} vertex of Y. Since f is an L-homomorphism from $G \times H^k$ to H, A_3 is a walk inside $L(Y)$ from a_3 to b_3. Since Y is an arbitrary oriented path, we conclude that $(a_3, b_3) \in L(x, y)$.

Now by applying the above argument and assuming $b_1 = b_2 = d$ we conclude that $(f(x; a^k, c), d) \in L(x, y)$. This would imply that $(c_1, d) \in L(x, y)$. By similar argument one can show that $(c, f(y; d^k, d')) \in L(x, y)$ (assuming $a_1 = a_2 = c$). Now by setting $a_1 = a_2 = f(x; a^k, c)$ and $b_1 = d$ and $b_2 = d'$ we conclude that $(f(x; a^k, c), f(y; d^k, d')) \in L(x, y)$. Therefore by continuing this argument (1) is proved.

Proof of (b) : By similar argument as in (1) we conclude that $(f(x; a^k, c), e_1) \in L(x, z)$ because $(c, e_1), (a, e_1) \in L(x, z)$. Moreover, if $(c'_1, e_1), (c'_2, e_1), \ldots, (c'_k, e_1) \in L(x, z)$ then by applying similar argument as in (1) we conclude that $(f(x; c'_1, c'_2, \ldots, c'_k), e_1) \in L(x, y)$. Since c_1 is obtained from $\widehat{f}_{x,\{a,c\}}$, we conclude that $(c_1, e_1) \in L(x, z)$. Analogously one can show that $(d_1, e_1), (d_1, e_2) \in L(y, z)$.

Proof of (c) : Let $Z = y_1, y_2, \ldots, y_n$ where $y_1 = y$ and $y_n = z$ be an arbitrary oriented path in G. Since $(d_1, e_1), (d_1, e_2) \in L(y, z)$, there exist two walks D_1, D_2 from d_1, d_1 to e_1, e_2 in $L(Z)$ that are congruent to Z. Now $f(y_n; r_p^k, s_p), \ldots, f(y_i; r_i^k, s_i), \ldots, f(y_1; r_1^k, s_1)$, $f(y; d, d, \ldots, d)$ is a walk from $e_3 = f(z; e_1^k, e_2)$ to e in $L(Z^{-1})$ (reverse of Z). Here r_i is the i^{th} vertex of D_1^{-1} and s_i is the i^{th} vertex of D_2. Therefore $(d_1, f(z; e_1^k, e_2)) \in L(y, z)$. $\diamond$

Claim 4.7 *In function Bi-Clique-Instance we have the following :*

α. *If $test_1$ is false then there is no homomorphism from G to H that maps x to c_1 and y to d_1 and z to e_1.*

β. *If $test_2$ is false then there is no homomorphism from G to H that maps x to c_1 and y to d_2 and z to e_1.*

γ. *Suppose the $test_1, test_2$ are both true. Suppose there exists a homomorphism ψ from G to H with $\psi(x) = c_1$ and $\psi(y) = d_1$ and $\psi(z) = e_1$. Then there exist a homomorphism ψ' from G to H with $\psi'(x) = c_1$ and $\psi'(y) = d$ and $\psi'(z) = e_1$ (here $d \in \widehat{f}_{y,\{d_1,d_2\}}$).*

Proof: Let $L_{x,c,y,d,z,e}$ be the subset of L where for every $z' \in V(G)$, $L_{x,c,y,d,z,e}(z') = \{e' | (c, e') \in L(x, z'), (d, e') \in L(y, z'), (e_1, e') \in L(z, z')\}$.

Let $G', L' \subseteq L_{x,c_1,y,d_1,z,e_1}$ be the instance in Symmetric-Difference$(G, L, f, x, c_1, y, d_1, d_2, z, e_1)$ in which $L'(x) = c_1$, $L'(y) = d_1$ and $L'(z) = e_1$.

Proof of (α, β) : Note that G' is an induced sub-digraph of G. The list L' is a proper subset of L because $L'(y)$ contains only d_1 and not d_2. If there exists a homomorphism g' from G' to H then by definition for every $v \in V(G')$, $g'(v) \in L'(v)$. This would mean by

induction hypothesis on the correctness of the entire Algorithm 4 that $flag[x, y, c_1, d_1](z, e_1)$ should be true, a contradiction.

Proof of (γ). Suppose $test_1$ is true. Suppose there exists a homomorphism $h_2 : G \to H$ with $h_2(x) = c_1$, $h_2(y) = d_2$, $h_2(z) = e_1$. Then we show that there exists a homomorphism $h_1 : G \to H$ with $h_1(x) = c_1$, $h_1(y) = d_1$, $h_1(z) = e_1$.

Let $G^r_L(y, d_1, d_2)$ be the induced sub-digraph of $G \times_L H^2$ with vertices (x_1, i_1, i_2) where $i_1, i_2 \in L(x_1)$ and for every vertex $j_1 \in L(x_1)$, $(d_2, j_1) \notin L(y, x_1)$. The arcs set of $G^r_L(y, d_1, d_2)$ is $\{(x_1, i_1, i_2)(y_1, j_1, j_2)|x_1y_1 \in A(G), i_1j_1, i_2j_2 \in A(H), i_1j_2 \notin A(H)\} \cup \{(x_1, i_1, i_2)(y_1, j_1, j_2)|y_1x_1 \in A(G), j_1i_1, j_2i_2 \in A(H), j_2i_1 \notin A(H)\}$.

We note that since $test_1$ is true, there exists a homomorphism g_1 from G' to H with $g_1(x) = c_1$, $g_1(y) = d_1$ and $g_1(z) = e_1$.

Case 1. First assume there is no $u \in V(G) \setminus \{x, y, z\}$ where $(d_1, e), (d_2, e) \in L(y, u)$. Let G_0 be the induced sub-digraph of G' with the vertices v where $(v, a', b') \in G^r_L(y, d_1, d_2)$.

Now add x and z into G_0. Define $h_1(v) = g_1(v)$ when $v \in V(G'_0)$. Note that in this case $G'_0 = G$ and hence h_1 is a homomorphism from G to H.

Let V be a bi-clique vector starting at x, y, z with c_1, d_1, d_2, e_1 for x, y, y, z. The next vertex of the bi-clique is $z_1 \in V(G)$ with vertices $a_1, b_1 \in L(z_1)$, (b_1 could be the same as a_1) where $(c_1, a_1), (c_1, b_1) \in L(x, z_1)$, $(d_i, a_1), (d_i, b_1) \in L(y, z_1)$, $1 \leq i \leq 2$. In general the $(\ell+3)^{th}$ vertex of V is a new vertex z_ℓ with the vertices $a_\ell, b_\ell \in L(z_\ell)$ such that $(a_j, a_\ell), (b_j, b_\ell) \in L(z_j, z_\ell)$, $j \leq \ell$, and also $(c_1, a_\ell), (c_1, b_\ell) \in L(x, z_\ell)$, $(d_1, a_\ell), (d_1, b_\ell), (d_2, a_\ell), (d_2, b_\ell) \in L(y, z_\ell)$ and $(e_1, a_\ell), (e_1, b_\ell) \in L(z, z_i)$. We also assume that $flag[x, y, c_1, d_1](z_\ell, a_\ell)$ is true if there is a choice.

Recall that we have assumed that $h_2(z) = e_1$ otherwise we handle the case when there is no intersection between g_1 and h_2. According to the SYMMETRIC-DIFFERENCE function we observe the following.

Observation 4.8 *Consider the lists $L(x, c_1, y, d_1, z, e_1)$. We may assume that for each $v \in V(G)$ there exists at least one $i \in L(x, c_1, y, d_1, z, e_1)(v)$ such that $flag[y, d_1](v, i)$ is true.*

Now we are going to define h_1 from G to H which is constructed piecewise. Let $C \subseteq V(G')$ consisting of vertices v' where there exists some element $i \in L'(v')$ such that $(i, a_1) \in L(v', z_1)$, and $flag[z, e_1](z_1, a_1) = true$, and $flag[x, y, c_1, d_1](z_1, a_1) = true$. This is because of the above Observations.

The case when $flag[x, y, c_1, d_1](z_1, a_1) = false$ is handled similarly. Recall that $a_1 \in L(z_1)$ is a fixed element and note that $(e_1, a_1) \in L(z, z_1)$ by the construction of V.

Case 2. Assume that $(d_1, h_2(z_1)) \notin L(y, z_1)$. Note that since $flag[x, y, c_1, d_1](z_1, a_1)$ and $flag[z, e_1](z_1, a_1)$ are true, we may assume the homomorphism g_1 is a homomorphism from G_0, the sub-digraph of G' induced by the vertices in C, to H. Note that now by definition of C

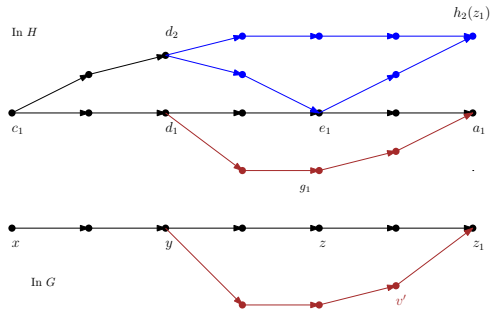

Figure 2: First illustration for proof of γ

we may assume that g_1 is in such a way that if $v' \in G'$ and $v'z_1 \in A(G)$ then $g_1(v')a_1 \in A(H)$. This can be assumed because in construction of V we assume that $flag[x, y, c_1, d_1](z_1, a_1)$ is true. Now add x and z into G_0. Define $h_1(v) = g_1(v)$ when $v \in V(G_0)$ (see Figure 2). Now if one of the following holds :

1. there is no other vertex in V, i.e. $\ell = 1$.
2. h_2 is in such a way that $(d_1, h_2(z_i)) \notin (y, z_i)$, $2 \leq i \leq \ell$.

Then we define h_1 for $G \setminus G_0$ to be g_1^1 : a homomorphism obtained from calling Symmetric-Difference$(G, L, f, z, e_1, z_1, a_1, h_2(z_1))$. Note that h_1 on G_0 is a homomorphism and h_2 on $G \setminus G_0$ is also a homomorphism. For the arcs that goes across G_0, $G \setminus G_0$, h_1 is a homomorphism because of the assumption on g_1.

If none of the $1, 2$ above holds we continue as follows. By induction hypothesis and because we have run Symmetric-Difference we may assume there exists a homomorphism g_1^1 on the digraph considered in Symmetric-Difference$(z, e_1, z_1, a_1, b_1, z_2, a_2)$. Here $a_2 = h_2(z_1)$ and $b_2 = h_2(z_2)$ (see Figure 3). Let G_1 be the induced sub-digraph of G with the vertices v such that $(v, g_1^1(v), h_2(v)) \in G^r_{L_1}(z_1, a_1, b_1)$. Here $L_1(v) = \{e' | (d_1, e') \in L_{x,c_1,y,d_1,z,e_1}(y, v)\}$. Define $h_1(v_1) = g_1^1(v_1)$ when $v_1 \in V(G_1)$. If for no other vertex w of G, w is in V then we define $h_1(w) = h_2(w)$ and as we argue above h_1 is a homomorphism from G to H. Otherwise we continue this way. Since we handle cut-vertices first, every vertex of G would be considered in this process and an in particular we would be considering vertex z_2 which plays the same role as z_1. As we continue we can partition the vertices of G into $G_0, G_1, \ldots, G_t$ where there exist homomorphisms g_1^i, $1 \leq i \leq t$ which are bases of the homomorphism h_1.

This would imply that in the construction of the lists in funvtion Bi-clique-Instace we may assume assume that d_2 is not included in the list of y.

Case 3. Assume that $(d_1, h_2(z_1)) \in L(y, z_1)$. According to the argument in Case 2 we may assume that homomorphism g_1 is in such a way that if $v' \in G'$ and $v'z_1 \in A(G)$ then $g_1(v')h_2(z_1) \in A(H)$.

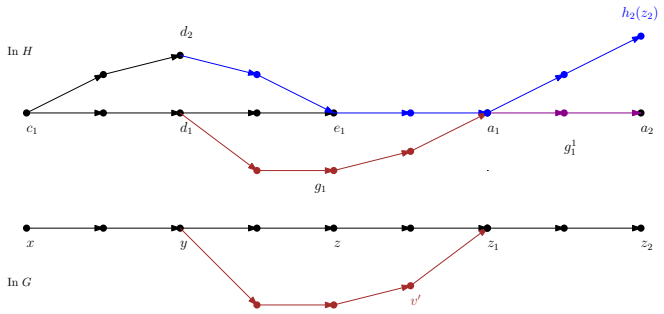

Figure 3: Second illustration for proof of γ.

Now add x and z into G_0. Define $h_1(v') = g_1(v')$ when $v' \in V(G_0)$. If for every vertex $u \in G \backslash G_0$, $(d_1, h_2(u)) \in L(y, u)$ then we set $h_1(u) = h_2(u)$. Note that h_1 on G_0 is a homomorphism and h_2 on $G \backslash G_0$ is also a homomorphism. For the arcs that goes across G_0 and $G \backslash G_0$, h_1 is a homomorphism because of the assumption on g_1. If there exists some vertex $u \in G \backslash G_0$ where $(d_1, h_2(u)) \notin L(z, u)$ then similar to Case 2 we define h_1 piecewise. This means we assume that $(e_1, h_2(z_2)) \notin L(z, z_2)$, and again we define part of h_1 according to the homomorphism g_1^2 which is from the instance Symmertic-Differece$(z, e_1, z_2, a_2, h_2(z_2), z_1, a_1)$ and continue as in Case 2.

Since $test_1, test_2$ are true, there exist homomorphisms g_1 and g_2 from G_1' to H and from G_2' to H where G_1', G_2' are induced sub-digraph of G in instances Symmetric-Difference for $G, L, f, x, c_1, y, d_1, d_2, z, e_1$ and Symmetric-Difference for $G, L, f, x, c_1, y, d_2, d_1, z, e_1$ respectively. By Claim 4.6 we conclude if g_1 and g_2 exist then the out-put of instance Symmetric-Difference$(G, L, f, x, c_1, y, d_3, d_1, z, e_1)$ where $d_3 = f(y, d_1^k, d_2)$, is a homomorphism g_3 that obtained as follows. For a vertex $v \in V(G')$, set $g_3(v) = f(v, g_1(v)^k, g_2(v))$. This would imply that we can replace $\widehat{f}_{y,\{d_1,d_2\}}$ with only one element $d \in \widehat{f}_{y,\{d_1,d_2\}}$ obtained by function Restricted-Instace (line 3). $\diamond$

Proof of 4 : In the Algorithm 4 we consider pairs $(x, a) \in V(G \times H)$ where $a \in L(x)$ and $\exists w_2 \in V(G \times H^k)$ with $f(w_1 = (x; a_1', a_2', \ldots, a_k')) = a$. Suppose the test $T_{x,c}$ (where $f(x; b^k, a) = c$) with respect to the lists L' succeeds, in other words, the L'-homomorphism g exists at some stage of the algorithm.

In the Update-F function for every $y \in V(G')$ and every $a_1 \in L(y)$, such that $(y, a_1, g(y))$ belongs to $G_L(x, a, c)$, the f value of $w_2 = (y; b_1', b_2', \ldots, b_k')$ with $f(w_2) = a_1$ is going to changed to some new value. Once the f value of some k-tuple in $L(y)$ changed from a_1 to something else, there would be no k-tuple in $L(y)$ that its value is set to a_1 in the further steps of the Algorithm. Moreover, for every $(a, e) \in L(x, y)$ the value of f for w_1, w_2 with $f(w_1) = a$ and $f(w_2) = e$ would change simultaneously.

We also note that if $flag[y, a_1]$ is set to be false inside the main loop (after it was considered

for a test) then it is not going to be considered in testing for another Small-Instance or Restricted-Instance.

Consider the case when we enp up having cut-vertices at each step of the Algorithm 4. We choose a cut vertex (x, a) and assign x to a. Now either this assignment would work out or it fails. If it fails then we remove (x, a) from $G \times_L H$ and hence we end up having connected components and according to Observation 3.7 we consider each valid connected component of $G \times_L H$ separately. Therefore the overall running time would be the sum of the running time of each connected components. This means that we partition the vertices of $G \times_L H$ into connected components. The work to do in each component would be modifying f which would takes $\mathcal{O}(|G||H|^k)$ because we potentially modify each k tuple inside the list of each vertex y of G. Therefore overall it takes $\mathcal{O}(|G|^3|H|^{k+2})$ if we end up having the L lists not weakly connected. Note that at the end we need to apply RemoveMinority algorithm which we assume there exists one with running time $\mathcal{O}(|G|^3|H|^3)$.

Now consider the case where we look at bi-cliques at some stage of the algorithm. We first perform Symmetric-Difference test. We consider triple $x, y, z \in V(G)$ and four vertices $c_1 \in L(x)$, $d_1, d_2 \in L(y)$, $e_1 \in L(z)$. The instance we construct in $I_1 = \textsc{Symmetric-Difference}(G, L, f, x, c_1, y, d_1, d_2, z, e_1)$ partition the lists L into two disjoint lists. $L_1(v) = \{i | (c, i) \in L(x, v), (d_1, i) \in L(y, v), (e_1, i) \in L(z, v), (y, d_2) \notin L(y, v)\}$. $L_2(v) = L(v) \setminus L_1(v)$. Now solving $I_2 = \textsc{Symmetric-Difference}(G, L, f,$
$x, c_1, y, d_2, d_1, z, e_1)$ uses a subset of L_2. Therefore the running time for I_1 and I_2 would be the summation of the tasks for two disjoint smaller instances. Now if we keep continue seeing cut-vertices in I_1 and I_2 then overall we would have a polynomial algorithm which in this case has running time $\mathcal{O}(|G|^6|H|^{k+2})$.

Let $c_1, c_2, \ldots, c_t \in L(x)$ and $d_1, d_2, \ldots, d_r \in L(y)$ such that they induce a bi-clique in L. Now consider an instance I_1 of the problem started at (x, c_1, y, d_1). If there does not exist z, $e_1 \in L(z)$ such that $(d_1, e_1) \in L(y, z)$ and $(d_2, e_1) \in L(y, z)$ then its means we should consider each connected component of the lists in instance I_1 which partition the tasks into disjoint sub-tasks. Thus we may assume that such z and $e_1 \in L(z)$ exist. According to the Algorithm 4 function Bi-Clique-Instances instead of $d_1, d_2, \ldots, d_k$ we only use one element d or we won't consider instance (x, c_1, y, d_1, z, e_1).

Let V be a bi-clique vector starting at x, y, z with c_1, d_1, d_2, e_1 for x, y, y, z and suppose d exists. Let I_1 be such an instance. The next vertex of the bi-clique is $z_1 \in V(G)$ with vertices $a_1, b_1 \in L(z_1)$, (a_1 could be the same as b_1) where $(c, a_1), (c, a_2) \in L(x, z_1)$, $(d_i, a_j) \in L(y, z_1)$, $1 \leq i, j, \leq 2$. In general the $(\ell + 3)^{th}$ vertex of V is a new vertex z_ℓ with the vertices $a_\ell, b_\ell \in L(z_\ell)$ such that $(a_j, a_\ell), (b_j, b_\ell) \in L(z_j, z_\ell)$, $j \leq \ell$, and moreover $(c, a_\ell), (c, b_\ell) \in L(x, z_\ell)$ and $(d_1, a_\ell), (d_1, b_\ell), (d_2, a_\ell), (d_2, b_\ell) \in L(y, z_\ell)$ and $(e_1, a_\ell), (e_1, b_\ell) \in L(z, z_i)$.

At each step we replace $\widehat{f}_{z_j, \{a_j, b_j\}}$ in $L(z_j)$ by one element b. Therefore the entire instance I_1 becomes just singleton elements at each vertex z_j in V.

However, there could be some other bi-clique of L on x, z with $c, c_1, c_2, \ldots, c_t \in L(x)$ and $e_1, e_2, \ldots, e_r \in L(z)$ and in this case the $flag[x, c](z, e_i)$, $1 \leq i \leq r$ may not be false. But this

means we have not reached (z, e_i) inside instance I_1, i.e. in the recursive call in I_1 starting at y we don't get into vertex z and $e_i \in L'(z)$. In other words, (x, c) is cut vertex and that contradicts the assumption that we have a weakly connected component in I_1.

Now suppose inside (L', f')=Small-Instance$(L, f, x, c_1, flag)$ there is no bi-clique. This means when we look at y for every pair of vertices $d_1, d_2 \in L'(y)$ with $f'(y; d'^k, d_1) = d_2$, there is no vertex z such that $(d_1, e_1), (d_2, e_1) \in L'(y, z)$ (here $e_1 \in L'(y)$). This means when we construct sub-instance inside (L', f') starting at $d_1 \in L'(y)$ we would have connected components.

Therefore the entire algorithm runs in $\mathcal{O}(|G|^6|H|^{k+3})$. This is because we consider every triple x, y, z and we run $\mathcal{O}(|A(G)||H|^{k+3})$ to create each instance.

Closing remark Once we change the value of $f(x; a_1, a_2, \ldots, a_k)$ from a to c then potentially we need to modify the value for $f(y; b_1, b_2, \ldots, b_k)$ from an out-neighbor of a, say a' in $L(y)$ to an out-neighbor of c. As far as the modifying f is concern it would yield the same result if we start from $(x; b, \ldots, a, b, \ldots, b)$, a is in the i^{th} coordinate. $\diamond$

4.3 Proof of Theorem 1.1

By Lemma 4.3 f is still a homomorphism from $G \times H^k \to H$ consistent with the lists L of G after Algorithm 4. Moreover, we preserve the existence of a homomorphism from G to H after Algorithm 4. Now we can apply Theorem 3.10. We observe that the running time of PreProcessing function is $\mathcal{O}(|G|^3|H|^2)$.

According to the proof of Lemma 4.3 (2) the running time of Algorithm 4 is $\mathcal{O}(|G|^6|H|^{k+3})$. The running time of Algorithm 3.2 ($\mathcal{O}(|G||A(G)||H|^{k+1})$. Therefore the running time of the Algorithm 1 is $\mathcal{O}(|G|^6|H|^{k+3})$.

5 New Minority Algorithm

We develops a direct algorithm to handle the minority case in our main algorithm. Our algorithm by itself is interesting because it won't use the Maltsev polymorphism and it decides whether there exists a homomorphism from G to H providing that there $G \times_L H$ has a Maltsev polymorphism.

Again the idea is similar to the one handling the RemoveNotMinority case. At each step we consider a vertex x of G and two vertices $a, b \in L(x)$ and try to eliminate one of the a, b from $L(x)$. To decide whether to remove a or b we construct a smaller instance of the problem with respect to a say (G', H, L') and solve this instance recursively. Roughly speaking the $L'(y)$ consists of the elements e where $(a, e) \in L(x, y)$ and $(b, e) \notin L(x, y)$.

At the end we have singleton lists and if there is a homomorphism from G to H with the singleton lists then success otherwise we report there is no homomorphism from G to H. Based on the existence of a L'-list homomorphism from G' to H we may decide to keep in $L(x)$ or remove a. Once we have solutions to all the small instances we construct

a homomorphism g from G to H using these small homomorphisms. We mainly use the following so called rectangle property of the instance.

Definition 5.1 *We say two vertices $a, b \in L(x)$ lies on a rectangle if there exists vertex $y \in V(G)$ and two vertices $c, d \in L(y)$ with congruent walks A_1, A_2, B_1, B_2 all in $L(Y)$ from a, b, a, b to c, d, d, c respectively. Here Y is an oriented path from x to y in G.*

Definition 5.2 *We say a, b are twin if they have the same in-neighbor and out-neighbor in every neighbor of x.*

We assume there is no twins in any $L(x)$. Otherwise we just simply remove one of the them.

Lemma 5.3 (rectangle-property) *Let X be an oriented path in G and let B, C, D be three walks in $L(X)$ all congruent to X where B is from a to c and C is from b to c and D is from b to D. Then there exists a walk E from a to d in $L(X)$ that is congruent with X.*

Proof: By following B, C, D on the vertices in X and applying the definition of polymorphism h, we conclude that E exists. ◇

Lemma 5.3 implies the following corollary.

Corollary 5.4 *If $(a, c) \in L(x, y)$ and $(b, c), (b, d) \in L(x, y)$ then $(a, d) \in L(x, y)$.*

Proof: The reason that $(a, c) \in L(x, y)$ is that for every oriented walk W from x to y in G there exists an oriented walk AC from a to c in $L(W)$. Now by assumption there exists walks BC, BD in $L(W)$ from b, b to c, d respectively that are congruent to W. Now by Lemma 5.3 there exists a walk in $L(W)$ from a to d which is congruent to W. Therefore $(a, d) \in L(x, y)$. ◇

$SD_L(x, a, b)$ **Symmetric-Difference-Construction** Let $SD_L(x, a, b)$ be a digraph with the vertices $\{(y, d, e) \mid (a, d), (b, e) \in L(x, y),\ (a, e), (b, d) \notin L(x, y)\}$. The arc set of $SD_L(x, a, b)$ is $A(SD_L(x, a, b)) = \{(y, d, e)(y', d', e') |\ yy' \in A(G), dd', ee' \in A(H), de', ed' \notin A(H)\} \cup \{(y, d, e)(y', d', e') |\ y'y \in A(G), d'd, e'e \in A(H), e'd, d'e \notin A(H)\}$. Moreover, every vertex of $SD_L(x, a, b)$ is reachable from (x, a, b). Note that (x, a, b) is a vertex of $SD_L(x, a, b)$.

$TR_L(x, a, b)$ **:** Let $TR_L(x, a, b)$ be an induced sub-digraph of $SD_L(x, a, b)$ with the vertices (y, d, e) such that for every $z \in V(G)$, $c \in L(z)$ with $(a, c), (b, c) \in L(x, z)$, if $(d, c) \in L(y, z)$ then $(e, c) \in L(y, z)$. Moreover, every vertex of $TR_L(x, a, b)$ is reachable from (x, a, b). Note that (x, a, b) is a vertex of $TR_L(x, a, b)$.

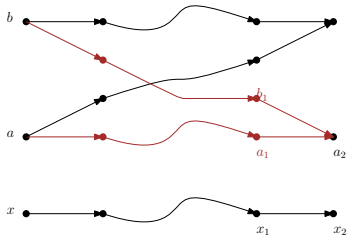

Figure 4: Component C shown by brown color

Strong component in $TR_L(x,a,b)$: Let C be an induced sub-digraph of $TR_L(x,a,b)$. We say C is a strong component of $TR_L(x,a,b)$ if for every $(x_1,a_1,b_1),(x_2,a_2,b_2)$ of C and for every $z \in V(G)$, $c \in L(z)$ with $(a,c),(b,c) \in L(x,z)$ then exactly one of the following happens (see Figure 4).

- $(a_1,c) \in L(x_1,z)$ and $(a_2,c) \in L(x_2,z)$
- $(a_1,c) \notin L(x_1,z)$ and $(a_2,c) \notin L(x_2,z)$.

Lemma 5.5 *The Algorithm 5 runs in $\mathcal{O}(|G|^3|H|^3)$. Moreover, if there is a homomorphism g from G to H with $g(x) \in \{a,b\}$ then there is a homomorphism from G to H after removing a or b from $L(x)$ according to Maltsev-(G,H,L,h).*

Proof: We first need to show that if we remove a vertex a' from $L(y)$, according to Algorithm 5 line (5) then the rectangle property is going to be preserved. For contradiction suppose this property does not preserve.

Thus there exits an oriented walk Y be in G from x_1 to x_2 going through y and there exist congruent walks A_1, B_1, A_2, B_2 in $L(Y)$ and congruent to Y with the following description. There exist $a_1, a_2 \in L(x_1)$ and $b_1, b_2 \in L(x_2)$ such that A_1 is in $L(Y)$ from a_1 to b_1 and A_2 from a_2 to b_2 and B_1 from a_1 to b_2 and B_2 from a_2 to b_1. Moreover, we may assume A_1 goes through a' before removing a' from $L(y)$. Let Z be an oriented path from x to y and then following Y from y. Since $a' \in L'(y)$ there exists a walk in $L(Z)$ from a to a'. Let b' be the next vertex after a' in $L(Z)$ and suppose yz is an arc of Z. Let b'' be the vertex before a' in A_1 and let z' be a vertex on Y before y. We may assume that b' stays in $L(z)$. This means $g^z_{a',c'}$ exists (here $c' \in A_2$ is the corresponding vertex to a'). Now if there is no path from a_1 to b' in $L(Z)$ which is congruent to portion of Z from x_1 to z and does not go through a' then we should have removed a_1 from $L(x_1)$ after doing the PreProcessing, a contradiction.

We use induction on the size of the input ($|G||H|$). The instance (G',L') constructed in line 5 of the Algorithm 5 is smaller than G and the list L' is also is a subset of L. We note that for every $y \in V(G')$ all the vertices $a' \in L'(y)$ such that $(a,a') \in L'(x,y)$ are in $L'(y)$. Therefore if G' does not admit a homomorphism to H then there is no homomorphism from G to H that maps x to a.

Algorithm 5 RemoveMinority – Using Matlsev Property

function RemoveMinority(G, H, L)
  for all $x \in V(G)$ s.t $|L(x) = 1|$ **do** $\psi(x) = L(x)$
  PreProcessing(G, L) and if a list becomes empty return $\emptyset$.
  If $G \times_L H$ is not connected then consider each valid connected component separately
  while $\exists x \in V(G), a, b \in L(x)$ where a, b lie on a rectangle **do**
    (G', L') = Symmetric-Difference(G, L, x, a, b).
    $g^x_{a,b}$ = RemoveMinority(G', H, L')
                                  ▷ $g^x_{a,b}$ is a L' homomorphism from G' to H
    if $g^x_{a,b}$ is empty **then**
      Remove a from $L(x)$.
      PreProcessing (G, H, L).
  Set $\psi(z) = c$ where $c \in L(z)$
  for all $x \neq z \in V(G)$ **do** set $\psi(x) = \emptyset$
  for all $y \in V(G)$ **do** $L'(y) = \{e|(c, e) \in L(z, y)\}$
  while $\exists x \in V(G)$ with $\psi(x) = \emptyset$ **do**
    Construct $TR_{L'}(x, a, b)$ for some $a, b \in L'(x)$
    Let C be a strong component of $TR_{L'}(x, a, b)$
    for all neighbors (y, a_1, b_1) of (x, a, b) inside C **do**
      Let C_1 be the set of vertices reachable from (y, a_1, b_1) inside C
      for all $y' \in V(G)$ s.t $(y', a'_1, b'_1) \in C_1$ **do**
        if $\psi(y') = \emptyset$ **then** set $\psi(y') = g^{y'}_{a_1,b_1}$
    Let $x_1x_2 \in A(G)$ $(x_2x_1 \in A(G))$, with $\psi(x_2) = \emptyset$.
    Let $(x_1, a_1, b_1) \in C$ with $\psi(x_1) = a_1$.
    Set $\psi(x_2) = a_2$ where $a_1a_2, b_1a_2 \in A(H)$ and $a_2 \in L'(x_2)$.
    for all $y \in V(G)$ **do**
      if $(a_2, e) \notin L'(x_2, y)$ **then** remove e from $L'(y)$.
  return ψ.

function Symmetric-Difference(G, L, x, a, b)
  Initiate empty lists L'.
  $L'(x) = \{a\}$.
  $L'(y) = \{d \ |(a, d) \in L(x, y), (b, d) \notin L(x, y), (y, d, e) \in SD_L(x, a, b)$ for some $e \in L(y)\}$
  Let G' be the induced sub-digraph of G with vertices y such that $L'(y) \neq \emptyset$.
  return (G', L')

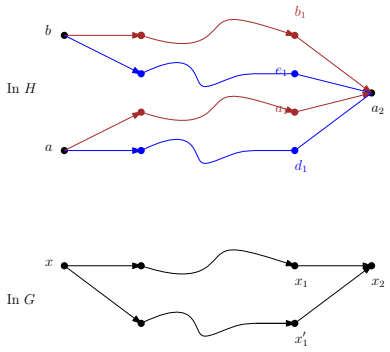

Figure 5: illustration of the proof

Suppose x_1x_2 is an arc of G. Suppose $(x_1, a_1, b_1) \in C$ and according to the Algorithm we may assume $\psi(x_1)$ is set to a_1.

First suppose $(x_2, a_2, b_2) \in C$ for some $a_2, b_2 \in L'(x_2)$. If both $(x_2, a_2, b_2), (x_1, a_1, b_1)$ are reachable from (y_1, d_1, e_1) (a neighbor of (x, a, b)) then in this case since $g^{y_1}_{d_1,e_1}$ exists, we have $\psi(x_1)\psi(x_2) = g^{y_1}_{d_1,e_1}(x_1)g^{y_1}_{d_1,e_1}(x_2) \in A(H)$. Note that since $(a, a_1) \in L'(x, x_1)$ and $(b, b_1) \in L'(x, y)$, and $x_1x_2 \in A(G)$, there exist $a'_2, b'_2 \in L'(x_2)$ such that $a_1a'_2, b_1b'_2 \in A(H)$. Now we may assume that $(x_2, a'_2, b'_2) \in C$ is not reachable (in $TR_{L'}(x, a, b)$) from (x_1, a_1, a_2). This means (x_2, a'_2, b'_2) is not adjacent to (x_1, a_1, b_1) (in $TR_{L'}(x, a, b)$) and we may assume $a_1a'_2, b_1a'_2 \in A(H)$.

Let (x'_1, d_1, e_1) be a vertex in C right before (x_2, a_2, b_2) on a path from (x, a, b) to (x_2, a_2, b_2) (see Figure 5). This means $x'_1x_2 \in A(G)$, $d_1a_2, e_1b_2 \in A(H)$, $d_1b_2 \notin A(H)$. Note that since (x'_1, d_1, e_1) and (x_1, a_1, b_1) belong to the same component C, by definition we have $d_1a'_2, e_1a'_2 \in A(H)$. Therefore by rectangle property we must have $d_1b_2 \in A(H)$, a contradiction that $(x'_1, d_1, e_1)(x_2, a_2, b_2)$ is an arc of $TR_{L'}(x, a, b)$.

Thus we may assume that $(x_2, a_2, b_2) \notin C$ for any $a_2, b_2 \in L'(x_2)$. This means we should have set $\psi(x_2) = a_2$ (according to line 23 of Algorithm 5) which is a neighbor of a_1. Note that for every other vertex $a'_2 \in L'(x_2)$ we have $a_1a'_2 \in A(H)$. Moreover, because of the property of C, for every vertex x'_1 if $(x'_1, d_1, e_1) \in C$ and $x'_1x_2 \in A(G)$ we have $d_1a_2 \in A(H)$. Since the list L' are going to prune so that they are consistent with $a_2 \in L(x_2)$, we can extend ψ to other vertices of G for which ψ has not been defined yet.

Running time The instance (G', L'_1) constructed from Symmetric-Difference(G, L, x, a, b) and instance (G', L'_2) constructed from Symmetric-Difference(G, L, x, b, a) are disjoint $(L'_1 \cap L'_2 = \emptyset)$. Moreover, we handle each connected component of $G \times_L H$ independently. Therefore the overall running time all the small instances (Symmetric-Difference) would be $\mathcal{O}(|G||H||A(G)||H^2|) = \mathcal{O}(|G|^3|H|^3)$. This is because we have $|G||H^2|$ instances and each of them takes $|A(G)|A(H)|$ to construct. The running time for finding ψ using the answers of the small instances is $|A(G)|A(H)|$ this is because we need to consider strong component

of $TR_L(x, a, b)$. Therefore the overall running time is $\mathcal{O}(|G|^3|H|^3)$. $\diamond$

Acknowledgements : We would like to thank Ross Willard and Pavol Hell for so many helpful discussions and their useful comment as well as their enormous support. We would like to thank Víctor Dalmau for so many useful questions and helpful comments. We would also like to thank Geoffrey Exoo, László Egri, for their comments.

References

[ABISV09] E. Allender, M. Bauland, N. Immerman, H. Schnoor, and H. Vollmer. The complexity of satisfiability problems: refining schaefer's theorem. *Journal of Computer and System Sciences*, 75(4): 245–254 (2009).

[BHM88] J. Bang-Jensen, P. Hell, G. MacGillivray. The complexity of colouring by semicomplete digraphs. *SIAM J. Discrete Math.*, 1 : 281–298 (1988).

[BH90] J. Bang-Jensen, P. Hell. The effect of two cyles on the complexity of colourings by directed graphs. *Discrete Appl. Math.*, 26 : 1–23 (1990).

[BKN09] L. Barto, Marcin Kozik, and Todd Niven. The CSP Dichotomy Holds for Digraphs with No Sources and No Sinks (A Positive Answer to a Conjecture of Bang-Jensen and Hell). *SIAM J. Comput*, 38(5) : 1782–1802 (2009).

[B02b] A.A. Bulatov. A dichotomy constraint on a three-element set. In *Proceedings of STOC* 649–658 (2002).

[B05] A.Bulatov. H-Coloring dichotomy revisited. *Theoret. Comp. Sci.*, 349 (1) : 31-39 (2005).

[B06] A. Bulatov. A dichotomy theorem for constraints on a three-element set. *Journal of the ACM*, 53(1): 66–120 (2006).

[B11] A.Bulatov. Complexity of conservative constraint satisfaction problems. *Journal of ACM Trans. Comput. Logic*, 12(4) : 24–66 (2011).

[B17] A dichotomy theorem for nonuniform CSPs. CoRR abs/1703.03021 (2017).

[BD06] A. Bulatov and V. Dalmau. A Simple Algorithm for Mal'tsev Constraints. *SIAM J. Comput.*, 36(1): 16–27 (2006).

[BJK05] A.A. Bulatov, P. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM journal on computing*, 34(3): 720-742 (2005).

[JB17] J. Bulin. Private communication.

[CCL13] J.Y. Cai, X. Chen, P. Lu. Graph Homomorphisms with Complex Values: A Dichotomy Theorem. *SIAM J. Comput.*, 42(3): 924–1029 (2013).

[CVK10] C. Carvalho, V.Dalmau, and A.A. Krokhin. CSP duality and trees of bounded pathwidth. *Theor. Comput. Sci.*, 411(34–36): 3188–3208 (2010).

[CKS01] N. Creignou, S. Khanna, and M. Sudan. Complexity Classifications of Boolean Constraint Satisfaction Problems. *SIAM Monographs on Discrete Math. and Applications*, vol. 7 (2001).

[CL14] P. Csikvári and Z. Lin. Graph homomorphisms between trees. *Elec. J. Combin.*, 21 : 4–9 (2014).

[D92] R. Dechter. Containt networks. *Encyclopedia of Artificial Intelligence* 276–285 (1992).

[D00] V. Dalmau. A new tractable class of constraint satisfaction problems. In *Proceedings 6th International Symposium on Artificial Intelligence and Mathematics*, 2000.

[DF03] V. Dalmau, D. Ford. Generalized satisfiability with k occurrences per variable: A study through delta-matroid parity. In *Proceedings of MFCS 2003, Lecture Notes in Computer Science*, 2747 : 358–367 (2003).

[EHLR14] L.Egri, P.Hell, B.Larose, and A.Rafiey. Space complexity of List H-coloring : a dichotomy. In *Proceedings of SODA*, (2014).

[F01] T. Feder. Homomorphisms to oriented cycles and k-partite satisfiability. *SIAM J. Discrete Math.*, 14 : 471–480 (2001).

[F06] T. Feder A dichotomy theorem on fixed points of several nonexpansive mappings. *SIAM J. Discrete Math.*, 20 : 291–301 (2006).

[FHH99] T.Feder, P.Hell, J.Huang. Bi-arc graphs and the complexity of list homomorphisms. *J. Graph Theory*, 42 : 61–80 (1999).

[FH98] T. Feder, P. Hell. List homomorphisms to reflexive graphs. *J. Comb. Theory Ser.*, B 72 : 236–250 (1998).

[FHH99] T. Feder, P. Hell, J. Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19 : 487–505 (1999).

[FHH03] T. Feder, P. Hell, J. Huang. Bi-arc graphs and the complexity of list homomorphisms. *J. Graph Theory*, 42 : 61–80 (2003).

[FHH07] T. Feder, P. Hell, J. Huang. List homomorphisms of graphs with bounded degrees. *Discrete Math.*, 307 : 386–392 (2007).

[FMS04] T. Feder, F. Madelaine, I.A. Stewart. Dichotomies for classes of homomorphism problems involving unary functions. *Theoret. Comput. Sci.*, 314 : 1–43 (2004).

[FV93] T.Feder and M.Vardi. Monotone monadic SNP and constraint satisfaction. In *Proceedings of STOC*, 612–622 (1993).

[FV98] T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28(1): 57–104 (1998).

[HN90] P. Hell and J. Nešetřil. On the complexity of H-colouring. *J. Combin. Theory B*, 48 : 92–110 (1990).

[HN04] P. Hell, J. Nešetřil. *Graphs and Homomorphisms*, Oxford University Press, 2004.

[HN08] P.Hell, J.Nešetřil. Colouring, Constraint Satisfaction, and Complexity. *Computer Science Review* , 2(3): 143–163 (2008).

[HR11] P.Hell and A.Rafiey. The Dichotomy of List Homomorphisms for Digraphs. In *Proceedings of SODA* 1703–1713 (2011).

[HR12] P. Hell and A. Rafiey. The Dichotomy of Minimum Cost Homomorphism Problems for Digraphs. *SIAM J. Discrete Math.*, 26(4): 1597–1608 (2012).

[KSDR96] L.G. Kroon, A. Sen, H. Deng, A. Roy. The optimal cost chromatic partition problem for trees and interval graphs. In *Graph-Theoretic Concepts in Computer Science (Cadenabbia, 1996), Lecture Notes in Computer Science*, 1197 : 279–292 (1997).

[K92] V. Kumar. Algorithms for constraint-satisfaction problems. *AI Magazine*, 13 :32–44 (1992).

[J98] P. Jeavons. On the Algebraic Structure of Combinatorial Problems. *Theor. Comput. Sci.*, 200(1-2): 185-204 (1998).

[L75] R. Ladner. On the Structure of Polynomial Time Reducibility. *Journal of the ACM (JACM)*, 22(1): 155–171 (1975).

[LZ03] B. Larose, L. Zádori. The complexity of the extendibility problem for finite posets. *SIAM J. Discrete Math.*, 17 : 114–121 (2003).

[MM08] M. Maroti, and R. McKenzie. Existence theorems for weakly symmetric operations. *Algebra Universalis*, 59 : 463–489 (2008).

[S78] T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings of STOC*, 216–226 (1978).

[S10] M.Sigge. A new proof of the H-coloring dichotomy. *SIAM J. Discrete Math.*, 23 (4) : 2204–2210 (2010).

[V00] M.Y. Vardi. Constraint satisfaction and database theory: a tutorial. *Proceedings of the 19th Symposium on Principles of Database Systems (PODS)*, 76–85 (2000).

[Z17] D.Zhuk. The Proof of CSP Dichotomy Conjecture. CoRR abs/1704.01914 (2017).