Isomorphism of (mis)labeled graphs

Pascal Schweitzer[1,⋆]

The Australian National University, Canberra, Australia.
pascal.schweitzer@anu.edu.au

Abstract. For similarity measures of labeled and unlabeled graphs, we study the complexity of the graph isomorphism problem for pairs of input graphs which are close with respect to the measure. More precisely, we show that for every fixed integer k we can decide in quadratic time whether a labeled graph G can be obtained from another labeled graph H by relabeling at most k vertices. We extend the algorithm solving this problem to an algorithm determining the number ℓ of vertices that must be deleted and the number k of vertices that must be relabeled in order to make the graphs equivalent. The algorithm is fixed-parameter tractable in $k + \ell$.
Contrasting these tractability results, we also show that for those similarity measures that change only by finite amount d whenever one edge is relocated, the problem of deciding isomorphism of input pairs of bounded distance d is equivalent to solving graph isomorphism in general.

1 Introduction

Given two graphs G and H, the graph isomorphism problem asks whether there exists an isomorphism from G to H. That is, the problem asks whether there exists an adjacency and non-adjacency preserving bijection from the vertices of G to the vertices of H. Graph isomorphism is a computational decision problem contained in NP, since the isomorphism represents a witness checkable in quadratic time. However, it is not known whether the problem is NP-complete and not known whether the problem is polynomial-time solvable (see [14] or [19] for an introduction to the problem).

Since graph isomorphism still has unknown complexity, researchers have considered the complexity of the isomorphism problem on subclasses of graphs. The isomorphism problem is for example isomorphism complete (i.e., polynomially equivalent to graph isomorphism) on bipartite graphs and regular graphs (see [24]). Among many algorithms that have been developed for specific graph classes, there is Luks' [16] polynomial-time algorithm for graphs of bounded degree, and polynomial-time algorithms for graphs of bounded genus developed by Filotti and Mayer [10] and by Miller [18]. For the known bounded degree algorithm and the known bounded genus algorithms, the degree of the polynomial bounding the running time increases with increasing parameter (i.e., they have a

⋆ Supported by the National Research Fund, Luxembourg, and cofunded under the Marie Curie Actions of the European Commission (FP7-COFUND).

running time of $\mathcal{O}(n^{f(k)})$). Algorithms with uniformly polynomial running time (i.e., having a running time of $\mathcal{O}(f(k) \cdot n^d)$ with d fixed) have only been devised for the parameters eigenvalue multiplicity [9], color multiplicity [12], feedback vertex set number [15], and rooted tree distance width [21]. In parametrized complexity theory such algorithms are called *fixed-parameter tractable*. See [8] or [11] for general parameterized complexity theory, and the introduction of [15] for a graph isomorphism specific overview.

In the context of the isomorphism problem, the subproblems that have traditionally been investigated, including all of the ones just mentioned, impose restrictions on both input graphs. In this paper we investigate the effect on the complexity when the input graphs are related to each other, i.e., when a certain similarity between the input graphs is required. This allows each input to be an arbitrary graph, but of course the input graphs are not independent. For any given similarity (or distance) measure we thus investigate:

What is the complexity of graph isomorphism when the input is restricted to be a pair of similar graphs with respect to a given measure?

The permutation distance. Throughout the paper, we always assume the input graphs are defined over the same vertex set. With regard to our question, we consider the following measure for labeled graphs: Two isomorphic graphs have a distance of at most k, if there exists a permutation of the vertices that is an isomorphism and permutes at most k vertices. For a pair of non-isomorphic graphs the distance is infinite.

The motivation for this definition is the following question: Suppose we are to decide whether a labeled graph G is isomorphic to a labeled blueprint graph H. A one by one comparison of the edges trivially determines whether they are isomorphic as labeled graphs. If however a small number of vertices (say k) in G have been mislabeled, how do we determine whether G matches the blueprint H, and how do we determine the vertices that have been mislabeled?

We show in Section 2 that there is a fixed-parameter tractable algorithm with a running time of $\mathcal{O}(f(k)n^2)$ for this problem. In other words, if the number of mislabeled vertices is bounded, an isomorphism can be found in time quadratic in the total number of vertices.

As a generalization, we show in Section 3 that the computation of the maximum common subgraph distance of a graph H to all graphs obtained from G by permuting at most k vertices is fixed-parameter tractable. That is, in the sense of the previous motivation, this extended problem asks: Given a labeled blueprint graph H and a labeled graph G in which not only k vertices have been mislabeled, but additionally for some vertices the adjacencies to other vertices have been altered, how do we detect the mislabeled vertices and the vertices for which the adjacencies have been altered?

Other similarity measures. In Section 4 we consider alternative similarity measures that have been defined for graphs in the literature. Formally, for a similarity measure d we denote by $d(G, H)$ the distance of G and H. Given a bound $k \in \mathbb{N}$

on the similarity, the question highlighted above asks for the complexity of graph isomorphism on input graphs G and H with $d(G, H) \leq k$.

There are two substantially different kinds of graph similarity measures, namely similarity measures that apply to labeled graphs and similarity measures that apply to unlabeled graphs. From a labeled distance measure one can obtain an unlabeled distance measure in the following general way: The (unlabeled) distance from G to H is the shortest labeled distance of G to some permutation of H. (Recall that we always assume that the input graphs are defined on the same vertex set and they thus in particular have equal size.)

Among the labeled (and thus also among the unlabeled) distance measures that have been considered are the maximum common subgraph and the maximum common contract distance, both first defined by Zelinka [22,23]. These distances measure the difference of the size of the input graphs to the size of the largest common subgraph and to the size of the largest common contract, respectively. Recall that a contract is a graph obtained by repeatedly contracting an edge, i.e., identifying two adjacent vertices. Further measures applicable to both labeled and unlabeled graphs are the edge slide distance (Johnsen [13]), the edge rotation distance (Chartrand, Saba, and Zou [4]) and with them also the edit distance (see [2]). These measures are based on local edge replacements according to certain allowed moves. Finally, the more recent cut distance [1] considers all bipartitions of the vertices and among those measures the maximum number of crossing edges contained in exactly one of the input graphs.

An intrinsically unlabeled distance measure is the spectral distance [20], which is defined as the square root of the sum of the squared differences of the eigenvalues of two graphs.

Using simple reductions, we show (Section 4) that for a large class of measures, including all of the above, the isomorphism problem of similar graphs is isomorphism complete, both in the labeled and the unlabeled case, except of course for the permutation distance treated in Section 2.

2 The permutation distance as similarity measure

We now consider the similarity measure which, for two labeled graphs over the same vertex set, measures how many vertices have to be permuted in order to obtain identical graphs. The distance of two non-isomorphic graphs is infinite.

For every fixed integer k, there is a trivial algorithm running in polynomial time $\mathcal{O}(k!\binom{n}{k}n^2) \subseteq \mathcal{O}(n^{k+2})$ that checks whether two graphs have distance at most k: Indeed, by testing all permutations of all k-tuples of vertices and checking whether the obtained graphs are identical, every isomorphism with the required properties will be found. Before we develop an $\mathcal{O}(n^2)$ algorithm we review basic definitions and notation for permutation groups.

Basic definitions and notation for permutation groups: Let π be a permutation of some finite set V. The *orbit* of an element $u \in V$ is the set $\{u, \pi(u), \pi^2(u), \ldots\}$. The *support* of π, denoted by $\mathrm{supp}(\pi)$, is the set $\{u \in V \mid \pi(u) \neq u\}$ of elements

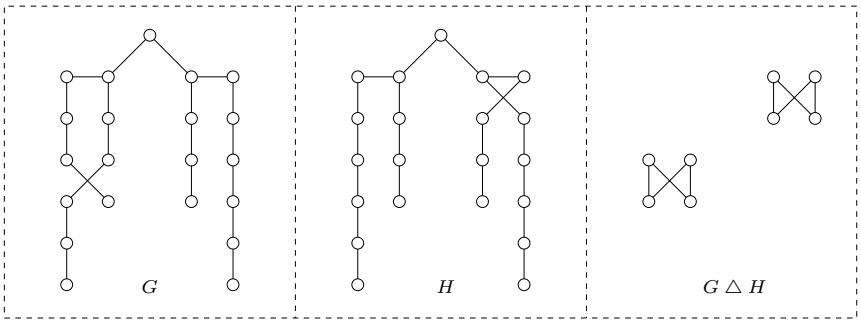

Fig. 1. Only one isomorphism exists between the depicted graphs G and H. This isomorphism fixes the vertex at the top and swaps the left part with the right part hanging from that vertex. The example shows that there does not necessarily exist an isomorphism from G to H that maps a vertex in $G \triangle H$ to a vertex in $G \triangle H$.

that are not fixed. The composition of two permutations $\pi \cdot \pi'$ is the permutation obtained by first applying π and then π'. The permutation π gives rise to a permutation on the set of all pairs of elements of V. For a pair of elements (u, v) we define $\pi((u, v))$ as $(\pi(u), \pi(v))$. Abusing notation we denote this as $\pi(u, v)$. For the purpose of this paper we define the complexity of a permutation.

Definition 1. *For a permutation π its* complexity $\mathrm{compl}(\pi)$ *is the size of the support of π minus the number of orbits of π which are of size at least 2.*

Observe that for any non-trivial permutation π it holds that $0 \leq \mathrm{compl}(\pi) < |\operatorname{supp}(\pi)|$. Furthermore, if $\mathrm{compl}(\pi) = 0$ then π is the identity. The complexity of a permutation is the minimum number of transpositions whose multiplication gives the permutation [7], but we will not need this fact.

To develop our algorithm, we use the symmetric difference as a concise way of addressing vertices and edges causing two labeled graphs to be non-identical.

Definition 2. *For two graphs $G = (V, E_G)$ and $H = (V, E_H)$ we define their* symmetric difference *to be the graph $G \triangle H := (V', E_G \triangle E_H)$, where V' is the set of vertices that are endpoints of edges in $E_G \triangle E_H$.*

Intuitively, $G \triangle H$ is the unique smallest graph (w.r.t. inclusion) that has an edge for all vertex pairs which are adjacent in exactly one of the graphs G and H.

Since we assume that the input graphs G and H are defined over the same vertex set V, every isomorphism from G to H is represented by a permutation π of V. (Abusing terminology, we use the terms interchangeably.) To design an algorithm with a polynomial running time whose degree is independent of k, we require a small candidate subset of vertices that should be permuted. Intuitively, such a candidate subset could be $V(G \triangle H)$, since for every edge in $G \triangle H$ one of its endpoints has to be permuted. However, $V(G \triangle H)$ may be of linear size,

Algorithm 1 Labeled isomorphism algorithm $\mathrm{ISO}_k(G, H, c, \psi)$

Input: Two labeled graphs G and H over a vertex set V, a recursion depth c for the algorithm and a permutation ψ of V that permutes at most k vertices.

Output: An isomorphism $\phi\colon G \to H$ that has complexity at most c so that the composition $\phi \cdot \psi$ permutes at most k vertices, or report **false**, if no such isomorphism exists. The algorithm may also report **false** if all isomorphisms ϕ with the described properties do not satisfy $|\mathrm{supp}(\psi) \cup \mathrm{supp}(\phi)| \leq k$.

compute $G \triangle H$
if $G \triangle H$ is empty **then**
  return $\phi = \mathrm{id}$ // where id is the identity
end if
if $c > 0$ **then** // compute set of candidates C
  if $|V(G \triangle H)| \leq 2k$ **then**
    $C \leftarrow V(G \triangle H)$
  else
    if $G \triangle H$ has vertex cover number greater than k **then**
      return false
    else
      compute a vertex cover C of $G \triangle H$ of size at most k
    end if
  end if
  for all $v_1, v_2 \in C$ **do** // try swapping all pairs from C
    $\psi' \leftarrow (v_1, v_2) \cdot \psi$
    form graph H' from H by swapping the neighbors of the vertices v_1 and v_2
    if ψ' permutes at most k vertices **then**
      call $\mathrm{ISO}_k(G, H', c-1, \psi')$ and record the result in ϕ'
      if $\phi' \neq$ **false then**
        return $\phi = \phi' \cdot (v_1, v_2)$
      end if
    end if
  end for
end if
return false

even for bounded k, and more importantly there are examples of isomorphic labeled graphs for which no isomorphism maps a vertex in $V(G \triangle H)$ to a vertex in $V(G \triangle H)$. An example of such a pair of graphs is shown in Figure 1.

Description of the algorithm: For a fixed integer k (the number of vertices that may be permuted) algorithm ISO_k takes as input two labeled graphs G and H, a recursion depth c and a permutation ψ that keeps track of the alterations that have been applied to H. The integer c is an upper bound on the complexity of the permutation representing the isomorphism. To determine whether there exists an isomorphism permuting at most k vertices, the algorithm is called as $\mathrm{ISO}_k(G, H, k, \mathrm{id})$ (i.e., c is set to k, and ψ is the identity). The algorithm either outputs an isomorphism that permutes at most k vertices, or reports with **false** that no such isomorphism exists. (See Algorithm 1.)

The algorithm first determines whether the two labeled input graphs are identical. If not, it computes the symmetric difference of the two input graphs, and then, depending on the number of vertices in the symmetric difference, performs one of two operations: If the number of vertices in the symmetric difference is at most $2k$, for all pairs of vertices in the symmetric difference, it calls itself after having transposed the two vertices in the graph H. Otherwise it tries to compute a vertex cover of the symmetric difference of size at most k and, if successful, performs the transposition with every pair of vertices in the vertex cover, before performing the recursive calls. Returning from a successful recursive call, the obtained isomorphism is altered by the transposition of the two vertices that have been swapped, and subsequently returned. If a call is unsuccessful, algorithm proceeds with the next pair. At any point in time the algorithm keeps track of the permutation that has been performed on graph H with the help of input parameter ψ. If this permutation moves more than k vertices, no recursive call is performed.

The intention not to return any isomorphism that moves more than k vertices complicates the intermediate calls. More precisely, it may be the case that the permutation ψ that has been performed, together with some permutation ϕ of complexity c yields an isomorphism $\psi \cdot \phi$ that permutes at most k vertices, but that the intermediate steps require to consider permutations that permute more than k vertices. If no isomorphism avoids this issue, the algorithm may output **false** independent of these existing isomorphisms. Intuitively, this situation only happens, when on a previous recursion level the algorithm moved away from the sought isomorphism by swapping a wrong pair of candidate vertices.

2.1 Correctness:

Since the algorithm keeps track of the permutation performed on H, any isomorphism permutes at most k vertices. By definition of the algorithm, it is immediate that it never returns a map that is not an isomorphism (i.e., it has no false positives).

To establish the correctness, we show that if there exists an isomorphism that permutes k vertices or less, it will be found within a recursion depth of k. By induction, a product of c transpositions has complexity at most c, thus the algorithm will not output an isomorphism of complexity larger than c.

We show that if an isomorphism exists that permutes k vertices or less with a complexity of c, then the algorithm will call itself on an instance for which an isomorphism of complexity at most $c-1$ exists that also permutes k vertices or less. This is ensured by identifying two vertices that lie in the same orbit, and by using a basic fact on permutations:

Lemma 1. *If π is a permutation and v, v' are distinct elements in an orbit of π, then for the permutation $\pi' = \pi \cdot (v, v')$ (i.e., π followed by the transposition of v and v') it holds that* $\mathrm{supp}(\pi') \subseteq \mathrm{supp}(\pi)$ *and* $\mathrm{compl}(\pi') < \mathrm{compl}(\pi)$.

Proof. Transposing v and v' either splits the orbit that contains both v and v' into two orbits, of which at least one has size larger than 1, or v and v' form a

complete orbit of π, in which case the support of the permutation decreases by two, while the number of orbits of size larger than 1 is only reduced by one. □

To identify two vertices lying in the same orbit we consider the symmetric difference $G \triangle H$ of the input graphs.

Lemma 2. *Let G and H be non-identical graphs on the same vertex set. If there is an isomorphism from G to H represented by a permutation π, then there exist two distinct vertices $v, v' \in G \triangle H$ contained in the same orbit under π.*

Proof. Suppose otherwise, i.e., that for every vertex $v \in G \triangle H$ no other vertex in the orbit of v is contained in $G \triangle H$. Let (u, v) be an edge in $G \triangle H$. This implies that u or v is not fixed. By assumption v and u are the only vertices in their respective orbits which are contained in $G \triangle H$.

W.l.o.g. suppose $(u, v) \in E(G)$ and therefore $(u, v) \notin E(H)$. Let i be the least positive integer for which $(\pi^i(u), \pi^i(v)) = (u, v)$. We now show by induction that for all $j \in \{0, \ldots, i-1\}$ the pair $\pi^j(u, v) = (\pi^j(u), \pi^j(v))$ is an edge in G: By assumption $\pi^0(u, v) = (u, v) \in E(G)$ and for $0 < j < i$ if $\pi^{j-1}(u, v)$ is an edge in G, then $\pi^j(u, v)$ is an edge in H, since π is an isomorphism from G to H. By the definition of i we know that $\pi^j(u)$ is different from u, and therefore not contained in $V(G \triangle H)$ or $\pi^j(v)$ is different from v, and therefore not contained in $V(G \triangle H)$. Either way $\pi^j(u, v)$ is not in $G \triangle H$. Thus the fact that $\pi^j(u, v)$ is an edge in H implies that it is also an edge in G.

Finally, since $\pi^{i-1}(u, v)$ is an edge in G and π is an isomorphism, $\pi^i(u, v) = (u, v)$ is an edge in H. This yields a contradiction. □

To address the issue that the set $G \triangle H$ may be of linear size, and thus too large in order to be a candidate set for vertices to be permuted, we consider a vertex cover of $G \triangle H$.

Lemma 3. *Suppose G and H are graphs on the same vertex set. If π is an isomorphism from G to H, then the support of π is a vertex cover of $G \triangle H$.*

Proof. Since π is an isomorphism, no edge $(u, v) \in E(G \triangle H)$ can be fixed by π. Thus for every edge (u, v), one of the vertices u, v is not fixed by π. □

The lemma implies that $G \triangle H$ has vertex cover number at most k if there is an isomorphism that maps G to H and leaves at most k vertices unfixed.

Lemma 2 shows that there are vertices in $G \triangle H$ that lie in the same orbit. However when the candidate set C is restricted to a vertex cover of $G \triangle H$, to apply Lemma 1 we require that two vertices from C lie in the same orbit. Such two vertices do not exist in general, as shown by the example in Figure 2. The next lemma helps us to circumvent this problem, by performing a case distinction depending on the size of $G \triangle H$ relative to $2k$.

Lemma 4. *Let G and H be two graphs on the same vertex set V and let C be a vertex cover of $G \triangle H$ of size at most k. Suppose there is an isomorphism from G to H represented by a permutation π that leaves at most k vertices unfixed. If every orbit of π contains at most one vertex from C, then $|V(G \triangle H)| \leq 2k$.*

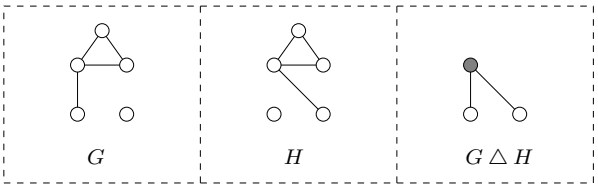

Fig. 2. The figure depicts isomorphic graphs G and H and their symmetric difference $G \triangle H$. In $G \triangle H$ the only minimum vertex cover C is highlighted by the gray shaded vertex. No isomorphism from G to H has an orbit that contains two vertices of C.

Proof. Since $|C| \leq k$ and π permutes at most k vertices, it suffices to show that π has no fix-point in $V(G \triangle H) \setminus C$. For contradiction assume $u \notin C$ is a vertex fixed under π and there is an edge $(u, v) \in G \triangle H$. W.l.o.g. assume $(u, v) \in E(G)$, and therefore $(u, v) \notin E(H)$. Since (u, v) cannot be fixed under π, the vertex v is in $C \cap \operatorname{supp}(\pi)$. Consider the orbit of (u, v) under π. Let i be the least positive integer for which $\pi^i(v) = v$. By assumption for no positive $j < i$ is $\pi^j(v)$ in C. Therefore, for $0 < j < i$, the pair $\pi^j(u, v) = (u, \pi^j(v))$ is not an edge in $G \triangle H$. By induction we show that for all $j \in \{0, \ldots, i-1\}$ the vertex pair $\pi^j(u, v)$ is an edge in G: This is true for $j = 0$, and if $\pi^{j-1}(u, v)$ is an edge in G, then since π is an isomorphism from G to H, $\pi^j(u, v)$ is an edge in H. Since $\pi^j(u, v)$ is not in $G \triangle H$ we conclude that $\pi^j(u, v)$ is an edge in G.

Finally since $\pi^{i-1}(u, v)$ is an edge in G and π is an isomorphism, $\pi^i(u, v)$ is an edge in H, which yields a contradiction. □

Assembling Lemmas 1–4 we now conclude that Algorithm 1 finds two vertices whose transposition reduces the complexity of the sought isomorphism.

Lemma 5. *Let ψ be a permutation. If two labeled graphs G and H differ by an isomorphism ϕ of complexity c such that $|\operatorname{supp}(\psi) \cup \operatorname{supp}(\phi)| \leq k$ then $\mathrm{ISO}_k(G, H, c, \psi)$ returns an isomorphism from G to H.*

Proof. We show the statement by induction on c. If $c = 0$ then G and H are identical and $\mathrm{ISO}_k(G, H, 0)$ returns the identity. Suppose $c > 0$. Let ϕ be the isomorphism that maps G to H and that fulfills the assumptions. If $|V(G \triangle H)| \leq 2k$ the algorithm simulates the permutation of all pairs of vertices in $V(G \triangle H)$ by swapping vertices in H. By Lemma 2 there exist two distinct vertices v_1, v_2 in $V(G \triangle H)$ that lie in the same orbit under ϕ. By Lemma 1 their transposition reduces the complexity of ϕ and does not increase the number of vertices that have to be permuted.

If on the other hand $|V(G \triangle H)| > 2k$, then by Lemma 3 the graph $G \triangle H$ has vertex cover number at most k. By Lemma 4, for any vertex cover C of $G \triangle H$ of size at most k there exist two vertices v_1, v_2 in C that lie in the same orbit. Again by Lemma 1 their transposition reduces the complexity.

Note that in both cases $v_1, v_2 \in \operatorname{supp}(\phi)$. Therefore $|\operatorname{supp}((v_1, v_2) \cdot \psi) \cup \operatorname{supp}(\phi \cdot (v_1, v_2))| \leq |\operatorname{supp}(\psi) \cup \operatorname{supp}(\phi)| \leq k$. Thus, by induction, the call to

$\mathrm{ISO}_k(G, H', c-1, \phi\cdot(v_1, v_2))$ with a permuted H returns an isomorphism. Since π is an isomorphism for the input pair (G, H) if and only if $\pi \cdot (v_1, v_2)$ is an isomorphism for the modified input pair (G, H'), the returned isomorphism is altered to an isomorphism from G to H and subsequently returned. □

The lemma establishes that the call $\mathrm{ISO}_k(G, H, k, \mathrm{id})$ of Algorithm 1 finds an isomorphism if there exists an isomorphism that permutes at most k vertices.

2.2 Running time:

Having established the correctness of the algorithm, we now analyze its running time. The following theorem shows that our problem is fixed-parameter tractable.

Theorem 1. *For any integer k Algorithm 1 solves graph isomorphism for labeled graphs that differ by a permutation of at most k vertices in $\mathcal{O}(f(k)n^2)$ time.*

Proof. The time spent within each call of the algorithm is dominated by the computation of $G \triangle H$ and the computation of a vertex cover of $G \triangle H$ of size up to k. The computation of $G \triangle H$ can be performed in $\mathcal{O}(n^2)$ time, as it only requires a simple one-by-one comparison of the edges in G and H. Having a representation of the graph $G \triangle H$, with the classical fpt-algorithms (see [11]), the vertex cover problem can then be solved in $\mathcal{O}(f(k)n^2)$ time.

It remains to bound the number of selfcalls of the algorithm. For this we observe that each iteration calls its own algorithm at most $(2k)^2$ times: Indeed this bound holds for both cases of the size of $V(G \triangle H)$. The recursion depth of the algorithm is at most k, thus there are at most $((2k)^2)^k$ calls of the algorithm. The overall running time is thus in $\mathcal{O}((2k)^{2k} \cdot f(k) \cdot n^2)$. □

Note that currently, for the parameter vertex cover number, the best known fpt-algorithm for the vertex cover problem, by Chen Kanj and Xia [5], runs in time $\mathcal{O}(1.2738^k + kn) \subseteq \mathcal{O}(1.2738^k \cdot n^2)$. This gives an overall running time of $\mathcal{O}((2k)^{2k} \cdot 1.2738^k \cdot n^2)$ for our algorithm.

3 Extension to maximum common subgraph distance

For two labeled graphs G and H defined on a vertex set of n vertices, let $d_0(G, H)$ be n minus the size of the maximum common (labeled) subgraph of G and H. That is, $d_0(G, H)$ is the number of vertices that have to be removed from the vertex set of the graphs G and H to obtain identical labeled graphs. It is easy to show that $d_0(G, H)$ is equal to the vertex cover number of $G \triangle H$. This implies that the computation of $d_0(G, H)$ is fixed-parameter tractable.

We define $d_k(G, H)$ to be the minimum of all $d_0(G', H)$ where G' ranges over all graphs obtained from G by permuting at most k vertices. For the complete graph K_n, the distance $d_0(G, K_n)$ is equal to n minus the size of the largest clique in G. Thus, with k as parameter, computation of $d_k(G, H)$ is W[1]-hard. (In fact, it is even NP-complete for $k = 0$.)

However, if we take both the parameters k and d_k for the input graphs into account, then the computation is fixed-parameter tractable.

Theorem 2. *There is an algorithm that computes for two graphs G and H the distance $d_k(G,H)$ (i.e., the maximum common subgraph distance between H and all graphs obtained from G by a permutation of at most k vertices) in time $\mathcal{O}(f(k,d_k(G,H))\cdot n^2)$. Here $f(k,d)=(k+d)^{2k}\cdot 2^{k\cdot(k+d)}$.*

The description of the algorithm, which is an adaptation of Algorithm 1, and the proof of the theorem, which generalizes the proof of Theorem 1, are deferred to the appendix.

4 Similarity measures and intractability

Contrasting the tractability of the isomorphism problem for graphs of low permutation distance, we now show intractability of the graph isomorphism problem for a wide range of other measures. The following folklore observation relates the evaluation of similarity measures to the complexity of similar graphs.

Theorem 3. *Let $d(\cdot,\cdot)$ be an arbitrary real-valued function that takes two labeled graphs as input.*

1. *If $d(G,H)\neq 0$ implies $G\ncong H$ and $d(G,H)=0$ can be decided in polynomial-time, then the problem of deciding isomorphism for graphs of bounded distance (i.e., $d(G,H)\leq k$ for a fixed constant k) is isomorphism complete.*
2. *If $d(G,H)=0$ is equivalent to $G\cong H$ then evaluation of $d(\cdot,\cdot)$ is at least as hard as deciding graph isomorphism.*

Proof. For the first part of the theorem observe that graph isomorphism reduces to the problem of deciding isomorphism of graphs of bounded distance in the following way: By assumption, it can be decided in polynomial time whether $d(G,H)=0$. If $d(G,H)>0$ then the graphs are non-isomorphic. Otherwise any algorithm that solves the isomorphism problem for graphs of bounded distance decides whether G and H are isomorphic.

The second claim follows from the fact that under the assumption, deciding $d(G,H)=0$ is equivalent to deciding whether G and H are isomorphic. □

The theorem applies to labeled similarity measures in general, but the assumptions are typically fulfilled by unlabeled similarity measures (i.e., measures invariant when replacing the inputs with isomorphic copies). For example, Part 1 applies to the spectral distance: Indeed the characteristic polynomial of both graphs can be computed in polynomial time, and their comparison determines whether the graphs have the same eigenvalues. This is of course well known (see [6, Section 1.2]). Part 2 applies to all other unlabeled measures from the introduction. The evaluation of many of them (e.g., rotation distance [17] and edit distance [3]) is even known to be NP-hard.

Independent of the complexity of the similarity measures' evaluation, we still obtain hardness results for similarity measures which govern graphs as close, whenever the graphs differ by the repositioning of one edge: We say that two graphs G and H over the same vertex set *differ by the repositioning of one edge*, if G and H have the same number of edges and $G\triangle H$ has exactly two edges.

Theorem 4. *Deciding whether two labeled graphs G and H that differ by the repositioning of one edge are isomorphic is graph isomorphism complete. Moreover, the problem remains graph isomorphism complete for input graphs that have a universal vertex (i.e., a vertex adjacent to every other vertex).*

Proof. We reduce the graph isomorphism problem to the problem of deciding whether two labeled graphs G and H that differ by the repositioning of an edge are isomorphic. Let G and H be two input graphs. W.l.o.g. we assume that G and H are connected, have the same number of edges, neither graph is complete, and their vertex-sets are disjoint. Let v and v' be two non-adjacent vertices of G. We construct a set of pairs of new input graphs as follows:

The first graph is always $U_1 = (G + (v, v')) \dot\cup H$, i.e., the disjoint union of the graph H with the graph obtained from G by adding the edge (v, v').

For the second graph we take all choices of non-adjacent vertices $u, u' \in H$ and form the graph $U_2(u, u') = G \dot\cup (H + (u, u'))$. Note that there are at most $|V(H)|^2$ such graphs. For any choice of non-adjacent vertices u, u' the graphs U_1 and $U_2(u, u')$ differ by at most two edges.

It suffices now to show that G and H are isomorphic if and only if there exist u, u' such that U_1 is isomorphic to $U_2(u, u')$. Suppose ϕ is an isomorphism from G to H, then by construction U_1 is isomorphic to $U_2(\phi(v), \phi(v'))$. Suppose now that G and H are non-isomorphic. Since G and H have the same number of edges, for any choice of u, u', any isomorphism must map the components of U_1 to components of $U_2(u, u')$. Moreover, due to the number of edges, the component of U_1 that is an exact copy of G must be mapped to the component of U_2 that is an exact copy of H, which yields a contradiction.

To see that the input graphs can also be required to have a universal vertex, note that the addition to both input graphs of a vertex that is adjacent to every other vertex preserves isomorphism, non-isomorphism and the symmetric difference. This operation thus reduces the problem to the special case of input graphs having a universal vertex. □

The theorem has intractability implications for all graph similarity measures which change only by a bounded amount whenever an edge is added or removed.

Corollary 1. *Let $d(\cdot, \cdot)$ be the labeled or unlabeled version of the maximum common subgraph distance, the maximum common contract distance, the edge rotation distance, the edge slide distance or the cut distance. There is a $k \in \mathbb{N}$, such that the graph isomorphism problem is graph isomorphism complete for the class of input pairs G and H with $d(G, H) \leq k$.*

Proof. By definition, for of each of the distances, graphs that differ by the repositioning of an edge and have a universal vertex are of bounded distance. □

Acknowledgements

I thank Danny Hermelin for posing to me the core question of the paper answered by Theorem 1. I also thank Reto Spöhel for helpful comments and suggestions.

References

1. C. Borgs, J. T. Chayes, L. Lovász, V. T. Sós, B. Szegedy, and K. Vesztergombi. Graph limits and parameter testing. In *STOC '06*, pages 261–270, New York, 2006.
2. H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.
3. H. Bunke. Error correcting graph matching: on the influence of the underlying cost function. *Pattern Analysis and Machine Intelligence*, 21(9):917–922, 1999.
4. G. Chartrand, F. Saba, and H.-B. Zou. Edge rotations and distance between graphs. *Časopis Pěst. Mat.*, 110(1):87–91, 1985.
5. J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In *MFCS '06*, pages 238–249, London, 2006.
6. D. M. Cvetković, P. Rowlinson, and S. Simić. *Eigenspaces of Graphs*. Cambridge University Press, Cambridge, UK, 1997.
7. J. Dénes. The representation of a permutation as the product of a minimal number of transpositions, and its connection with the theory of graphs. *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, 4:63–71, 1959.
8. R. G. Downey and M. R. Fellows. *Parameterized Complexity (Monographs in Computer Science)*. Springer, London, UK, 1998.
9. S. Evdokimov and I. N. Ponomarenko. Isomorphism of coloured graphs with slowly increasing multiplicity of Jordan blocks. *Combinatorica*, 19(3):321–333, 1999.
10. I. S. Filotti and J. N. Mayer. A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus. In *STOC '80*, pages 236–243, 1980.
11. J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer, London, UK, 2006.
12. M. L. Furst, J. E. Hopcroft, and E. M. Luks. Polynomial-time algorithms for permutation groups. In *FOCS '80*, pages 36–41, Washington, USA, 1980.
13. M. Johnson. An ordering of some metrics defined on the space of graphs. *Czechoslovak Math. J.*, 37(112)(1):75–85, 1987.
14. J. Köbler, U. Schöning, and J. Torán. *The graph isomorphism problem: its structural complexity*. Birkhäuser Verlag, Basel, Switzerland, 1993.
15. S. Kratsch and P. Schweitzer. Isomorphism for graphs of bounded feedback vertex set number. In *SWAT '10*, pages 81–92. Springer, 2010.
16. E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.
17. D. Marcu. Note on the edge rotation distance between trees. *International Journal of Computer Mathematics*, 30(1):13–15, 1989.
18. G. L. Miller. Isomorphism testing for graphs of bounded genus. In *STOC '80*, pages 225–235, New York, 1980.
19. P. Schweitzer. *Problems of unknown complexity : graph isomorphism and Ramsey theoretic numbers*. Phd thesis, Universität des Saarlandes, Saarbrücken, 2009.
20. R. C. Wilson and P. Zhu. A study of graph spectra for comparing graphs and trees. *Pattern Recognition*, 41(9):2833–2841, 2008.
21. K. Yamazaki, H. L. Bodlaender, B. de Fluiter, and D. M. Thilikos. Isomorphism for graphs of bounded distance width. *Algorithmica*, 24(2):105–127, 1999.
22. B. Zelinka. On a certain distance between isomorphism classes of graphs. *Časopis Pěst. Mat.*, 100(4):371–373, 1975.
23. B. Zelinka. Contraction distance between isomorphism classes of graphs. *Časopis Pěst. Mat.*, 115(2):211–216, 1990.
24. V. N. Zemlyachenko, N. M. Korneenko, and R. I. Tyshkevich. Graph isomorphism problem. *Journal of Mathematical Sciences*, 29(4):1426–1481, 1985.

A Appendix

The appendix contains a proof of Theorem 2. The proof consists of a description the algorithm that solves the extension of our problem to the maximum common subgraph distance, a correctness proof and a running time analysis for this algorithm.

A.1 Description of the algorithm for the extended problem:

For the theorem, we design an algorithm that is similar to Algorithm 1. The major modifications are as follows: Instead of checking whether $G \triangle H$ is empty in Line 2 we output the vertex cover number of $G \triangle H$. (This is necessary, since the algorithm does not know when it has found the permutation that minimizes the distance.)

Also, after computing a vertex cover C that is not too large, we iterate over all subsets $P \subseteq C$, and form a candidate set C' in a way depending on whether $(G \setminus P) \triangle (H \setminus P) > k + 2|C|$. Intuitively, the set P is the set of those vertices in C whose adjacencies have been compromised. Thus, for our purpose, it is not important whether adjacencies with vertices of P are preserved, when considering the symmetric difference. We therefore only consider vertices outside of P. However, there may be further vertices outside of C with compromised adjacencies. We take this into account when considering the size of $(G \setminus P) \triangle (H \setminus P)$ by comparing it to $k + 2|C|$ as opposed to $2k$.

Details are given by Algorithm 2.

A.2 Correctness of Algorithm 2:

The proof is based on the proof of Theorem 1. Appropriate generalizations of Lemmas 2–4 are assembled analogously to Lemma 5. The running time analysis of Algorithm 2 is an adaptation of the one in Section 2.

We need two more definitions concerning permutations: Let π be a permutation of a set V. For a set $C \subseteq V$, we define $\mathrm{Orb}_\pi(C)$ to be the set of elements that are contained in the same orbit as some element of C. For a graph $G = (V, E)$ with vertex set V we define $\pi(G)$ to be the graph $(V, \pi(E))$, where $\pi(E) = \{\pi(u, v) \mid (u, v) \in E\}$.

We first present three adequate adaptations/generalizations of Lemmas 2–4 given by Lemmas 6–8 respectively.

For the two input graphs G and H, our goal is to find a permutation π of at most k vertices that minimizes $d_0(\pi(G), H)$, the number of vertices that have to be deleted from $\pi(G)$ and from H to obtain identical labeled graphs. Thus, different from the special case of Algorithm 1 where $d_0(\pi(G), H) = 0$, there are edges in $G \triangle H$ for which no end-point has to be moved. However, according to the following lemma, we can still find two distinct vertices v and v' in $G \triangle H$ that lie in the orbit. Moreover, we can choose v and v' from a restricted subset of $G \triangle H$, which will later help us to take care of the vertices with compromised adjacencies.

Algorithm 2 Labeled isomorphism algorithm Ext-ISO$_k(G, H, c, d, \psi)$

Input: Two labeled graphs G, H over a vertex set V, a recursion depth c for the algorithm, a maximum distance d and a permutation ψ of V that permutes at most k vertices.

Output: The algorithm prints numbers. If there is a permutation π of complexity at most c such that $\pi \cdot \psi$ permutes at most k vertices for which $d_0(\pi(G), H) \leq d$, then the smallest printed number is the minimum of $d_0(\pi'(G), H) \leq d$ over all such permutations π'. Otherwise all printed numbers are larger than d. If no permutation π with the prescribed properties fulfills $|\operatorname{supp}(\psi) \cup \operatorname{supp}(\pi)| \leq k$, all printed numbers may also be larger than d.

compute $G \triangle H$
if $G \triangle H$ has vertex cover number greater than $k + d$ **then**
  return
end if
compute a smallest vertex cover C of $G \triangle H$
print $|C|$
if $c > 0$ **then** // compute set of candidates C'
  for all $P \subseteq C$ **do**
    $C' \leftarrow C$
    if $(G \setminus P) \triangle (H \setminus P)$ has at most $k + 2|C|$ vertices **then**
      $C' \leftarrow P \cup V((G \setminus P) \triangle (H \setminus P))$
    end if
    for all $v_1, v_2 \in C'$ **do** // try swapping all pairs from C'
      $\psi' \leftarrow (v_1, v_2) \cdot \psi$
      form the graph H' from H by swapping the neighbors of the vertices v_1 and v_2
      if ψ' permutes at most k vertices **then**
        call Ext-ISO$_k(G, H', c - 1, d, \psi')$
      end if
    end for
  end for
end if

Lemma 6. *Let G and H be two graphs on the same vertex set. If there is a permutation π of the vertices of G for which $d_0(\pi(G), H) < d_0(G, H)$, then there exist two distinct vertices $v, v' \in G \triangle H$ which are contained in the same orbit under π.*

Moreover, let C be a minimum vertex cover of $\pi(G) \triangle H$. If P is some subset of $\operatorname{Orb}_\pi(C) \cap V(G \triangle H)$, then v and v' can be chosen in $P \cup V((G \setminus P) \triangle (H \setminus P))$.

Proof. The first part of the theorem follows directly from the second by taking $P = \{\}$.

For a subset P of the vertices we define $P' = P \cup V((G \setminus P) \triangle (H \setminus P))$. To prove the second part, suppose for some $P \subseteq \operatorname{Orb}_\pi(C) \cap V(G \triangle H)$ the statement is not true, i.e., that for every vertex $v \in P'$ no other vertex in the orbit of v is contained in P'. Since C is a minimum vertex cover of $\pi(G) \triangle H$ and $d_0(\pi(G), H) < d_0(G, H)$, no vertex cover of $G \triangle H$ has size at most $|C|$.

Let $C' := \mathrm{Orb}_\pi(C) \cap P'$ be the set that consists of the vertices in P and those vertices in $(G \setminus P) \triangle (H \setminus P)$ which are contained in the same orbit as some vertex of C. Since there is only one vertex of P' in every orbit (because we suppose for P the theorem is not true), we know that $|C'| \leq |C|$. Thus, C' is not a vertex cover of $G \triangle H$ (and therefore not a vertex cover of P') and there is an edge (u, v) in $V((G \setminus P) \triangle (H \setminus P))$ with $u \notin C'$ and $v \notin C'$. This implies that u or v is not fixed. By assumption v and u are the only vertices in their respective orbits which are contained in P' and no vertex in their respective orbits is in C.

(Now, we are in a similar situation as in the proof of Lemma 2, hence a similar argument applies:)

W.l.o.g. suppose $(u, v) \in E(G)$ and therefore $(u, v) \notin E(H)$. Let i be the least positive integer for which $(\pi^i(u), \pi^i(v)) = (u, v)$. We now show by induction that for all $j \in \{0, \ldots, i-1\}$ the pair $\pi^j(u, v) = (\pi^j(u), \pi^j(v))$ is an edge in G: By assumption $\pi^0(u, v) = (u, v) \in E(G)$ and for $0 < j < i$ if $\pi^{j-1}(u, v)$ is an edge in G, then $\pi^j(u, v)$ is an edge in $\pi(G)$. But neither $\pi^j(u)$ nor $\pi^j(v)$ is in C, thus $\pi^j(u, v) \notin \pi(G) \triangle H$, and therefore $\pi^j(u, v)$ is an edge in H. By the definition of i either $\pi^j(u)$ is different from u, and therefore not contained in P' or $\pi^j(v)$ is different from v, and therefore not contained in P'. Either way $\pi^j(u, v)$ is not in $G \triangle H$. Thus the fact that $\pi^j(u, v)$ is an edge in H implies that it is also an edge in G.

Finally, since $\pi^{i-1}(u, v)$ is an edge in G, we know $\pi^i(u, v) = (u, v)$ is an edge in $\pi(G)$, and (again), since neither $\pi^j(u)$ nor $\pi^j(v)$ is in C, we conclude that (u, v) is an edge in H. This yields a contradiction. □

An analog of Lemma 3 takes into account that in order to cover $G \triangle H$ we also have to cover the edges $G \triangle H$ for which neither endpoint is moved.

Lemma 7. *Suppose G and H are graphs on the same vertex set. If π is a permutation of at most k vertices, then $G \triangle H$ has a vertex cover of size at most $d_0(\pi(G), H) + k$.*

Proof. Let C be a minimum vertex cover of $\pi(G) \triangle H$, which is necessarily of size $d_0(\pi(G), H)$. It suffices to argue that the set $C \cup \mathrm{supp}(\pi)$ is a vertex cover: For every edge $(u, v) \in E(G \triangle H)$ one of the vertices is not fixed, or $\pi(u, v) = (u, v) \in E(\pi(G) \triangle H)$. In both cases the edge is covered by a vertex in $C \cup \mathrm{supp}(\pi)$. □

After removing vertices whose adjacencies have been compromised we can again show that if no two vertices in a vertex cover lie in the same orbit under π then the size of the entire symmetric difference is bounded.

Lemma 8. *Let G and H be two graphs on the same vertex set V and let C be a vertex cover of $G \triangle H$. Suppose π is a permutation with $d_0(\pi(G), H) < d_0(G, H)$ that leaves at most k vertices unfixed.*

If in every orbit of π there is at most one vertex from C, then there is a subset $P \subseteq C$ such that $(G \setminus P) \triangle (H \setminus P)$ has at most $k + 2|C|$ vertices.

Proof. Let C' be a minimum vertex cover of $\pi(G) \triangle H$. Since $d_0(\pi(G), H) < d_0(G, H)$, we have $|C'| \leq C$.

Let $P := \mathrm{Orb}_\pi(C') \cap C$ be the set of those vertices in C which are contained the same orbit as some vertex of C'.

It suffices to show that, under the assumption, the permutation π has no fix-point in the set $V((G \setminus P) \triangle (H \setminus P)) \setminus (C \cup C')$.

Thus, suppose $u \in V((G \setminus P) \triangle (H \setminus P)) \setminus (C \cup C')$ is a vertex fixed under π. Then there is an edge $(u, v) \in V((G \setminus P) \triangle (H \setminus P))$. Since C is a vertex cover of $V(G \triangle H)$ and $u \notin C$ we know $v \in C \cap V(G \triangle H)$.

By the choice of u we have $u = \pi(u) \notin C'$. If v were fixed, then $(u, v) = \pi(u, v) \in \pi(G) \triangle H$. But since $\pi(u) \notin C'$ this would imply $\pi(v) \in C'$, which is not the case since $v \notin P$. Thus v is not fixed.

(Now, we are in a similar situation as in the proof of Lemma 4, hence a similar argument applies:)

W.l.o.g. assume $(u, v) \in E(G)$, and therefore $(u, v) \notin E(H)$. Consider the orbit of (u, v) under π. Let i be the least positive integer for which $\pi^i(v) = v$. By assumption for no positive $j < i$ is $\pi^j(v)$ in P. Therefore, for $0 < j < i$, the pair $\pi^j(u, v) = (u, \pi^j(v))$ is not an edge in $\pi(G) \triangle H$. By induction we show that for all $j \in \{0, \ldots, i-1\}$ the vertex pair $\pi^j(u, v)$ is an edge in G: This is true for $j = 0$, and if $\pi^{j-1}(u, v)$ is an edge in G, then $\pi^j(u, v)$ is an edge in $\pi(G)$. But neither $\pi^j(u) = u$ nor $\pi^j(v)$ is in C', thus $\pi^j(u, v) \notin \pi(G) \triangle H$, and therefore $\pi^j(u, v)$ is an edge in H. Since $\pi^j(u, v)$ is not in $G \triangle H$ we conclude that $\pi^j(u, v)$ is an edge in G.

Finally (again) since $\pi^{i-1}(u, v)$ is an edge in G, $\pi^i(u, v)$ is an edge in $\pi(G)$. Since neither $\pi^i(u) = u$ nor $\pi^i(v) = v$ is in C' we conclude that $\pi^i(u, v)$ is an edge in H, which yields a contradiction. □

Assembling Lemmas 6–8 we can show that in each iteration Algorithm 2 finds two vertices that lie in the same orbit of the sought permutation π that minimizes $d_0(\pi(G), H)$ and therefore will eventually print $d_0(\pi(G), H)$.

Lemma 9. *Let c,k,d be non-negative integers and let G and H be two labeled graphs on the same vertex set. If $d_k(G, H) \leq d$ then $d_k(G, H)$ is the smallest number printed by* Ext-ISO$_k(G, H, k, d, id)$. *If $d_k(G, H) > d$, all printed numbers are larger than d.*

Proof. The algorithm only recurses if the total permutation ψ performed on graph H moves at most k vertices. Thus, when printing in Line 6, the algorithm prints the number $d_0(\pi(G), H) \leq d$ for some permutation π that permutes at most k vertices. Therefore the algorithm does not output any number smaller than $d_k(G, H)$. It suffices now to show the following:

If π is a permutation of complexity at most c such that $\pi \cdot \psi$ permutes at most k vertices for which additionally $d_0(\pi(G), H) \leq d$ and $|\mathrm{supp}(\psi) \cup \mathrm{supp}(\pi)| \leq k$, then Ext-ISO$_k(G, H, c, d, \psi)$ prints a number $d' \leq d_0(\pi(G), H)$.

We show the statement by induction on c. For $c = 0$ the permutation π is the identity and if $d_0(G, H) \leq d$ then $G \triangle H$ has vertex cover number at most d thus the call Ext-ISO$_k(G, H, 0, d)$ will proceed to Line 6 and print $d_0(G, H)$.

Now suppose $c > 0$: If $d_0(\pi(G), H) \leq d$, then by Lemma 7, the graph $G \triangle H$ has vertex cover number at most $d_0(\pi(G), H) + k \leq d + k$. Thus Algorithm 2 will proceed up to Line 7.

If $d_0(\pi(G), H) > d_0(G, H)$ then the number d' printed in Line 6 is $d' = d_0(G, H) < d_0(\pi(G), H)$, which was to be shown. Otherwise, if $d_0(\pi(G), H) \leq d_0(G, H)$ then there are two cases:

Case 1): There are two vertices v_1 and v_2 in C which are contained in the same orbit under π. In this case, by Lemma 1, for $P = \{\}$ the transposition of v_1 and v_2 will reduce the complexity of π. Since $v_1, v_2 \in \operatorname{supp}(\pi)$, $|\operatorname{supp}((v_1, v_2) \cdot \psi) \cup \operatorname{supp}(\pi \cdot (v_1, v_2))| \leq |\operatorname{supp}(\psi) \cup \operatorname{supp}(\pi)| \leq k$.

Thus by induction the call Ext-ISO$_k(G, H', c-1, d)$, where H' is the graph obtained from H by transposing v_1 and v_2, will return a number $d' \leq d_0(\pi(G), H)$.

Case 2): In any orbit under π there is at most one vertex of C. Then by Lemma 8 there is a subset $P \subseteq C$ such that $(G \setminus P) \triangle (H \setminus P)$ has at most $k+2|C|$ vertices.

By Lemma 6 there are two vertices in $(G \setminus P) \triangle (H \setminus P)$ lying in the same orbit under π. The algorithm swaps these two vertices in H to obtain the graph H', decreasing the complexity, and calls itself on Ext-ISO$_k(G, H', c-1, d)$. As in the previous case, since $v_1, v_2 \in \operatorname{supp}(\pi)$, $|\operatorname{supp}((v_1, v_2) \cdot \psi) \cup \operatorname{supp}(\pi \cdot (v_1, v_2))| \leq |\operatorname{supp}(\psi) \cup \operatorname{supp}(\pi)| \leq k$. By induction, this call then returns a number $d' \leq d_0(\pi(G), H)$. □

A.3 Running time of Algorithm 2:

Theorem 5. *For non-negative integers k and d Algorithm 2 (which is called as* Ext-ISO$_k(G, H, k, d)$*) runs in $\mathcal{O}(f(k,d) \cdot n^2)$, where $f = (k+d)^{2k} \cdot 2^{k \cdot (k+d)}$.*

Proof. The time spent within each call of the algorithm consists of four parts: the computation of $V(G \triangle H)$, the computation of a vertex cover of $V(G \triangle H)$ of size up to $k+d$, a loop over all subsets $P \subseteq C$ and the recursive sub-calls. The first two parts can be performed in $\mathcal{O}(n^2)$. For the third part, each iteration for a subset P takes time $\mathcal{O}(|C|^2 n^2)$. Thus, for the third part in total, this gives a running time of $\mathcal{O}(|C|^2 2^{|C|} n^2)$. Since $|C|$ is bounded by $k+d$, this running time can be expressed as $\mathcal{O}(f'(d,k)n^2)$ for the function $f'(k,d) = (k+d)^2 \cdot 2^{k+d}$.

It remains to bound the number of selfcalls of the algorithm. For this we observe that each iteration recursively calls the algorithm at most $f'(d,k)$ times. The recursion depth of the algorithm is at most k, thus there are at most $f'(d,k)^k$ calls of the algorithm. For the function $f(k,d) = (k+d)^{2k} \cdot 2^{k \cdot (k+d)}$, the overall running time can thus be bounded by $\mathcal{O}(f(k,d) \cdot n^2)$. □