Support Computation for Mining Frequent Subgraphs in a Single Graph

Mathias Fiedler and Christian Borgelt
European Center for Soft Computing
c/ Gonzalo Gutiérrez Quirós s/n, 33600 Mieres, Spain
Email: mail@mathias-fiedler.info, christian.borgelt@softcomputing.es

***Abstract*—Defining the support (or frequency) of a subgraph is trivial when a database of graphs is given: it is simply the number of graphs in the database that contain the subgraph. However, if the input is one large graph, it is surprisingly difficult to find an appropriate support definition. In this paper we study the core problem, namely overlapping embeddings of the subgraph, in detail and suggest a definition that relies on the non-existence of equivalent ancestor embeddings in order to guarantee that the resulting support is anti-monotone. We prove this property and describe a method to compute the support defined in this way.**

I. Introduction

In recent years frequent subgraph mining has received intense and still growing attention, since it has a wide and constantly expanding range of applications areas, which include biochemistry, web mining, and program flow analysis. As a consequence, several frequent subgraph mining algorithms have been developed. Some of them rely on principles from inductive logic programming and describe the graph structure by logical expressions [5]. However, the vast majority transfers techniques that were originally developed for frequent item set mining. Examples include MolFea [8], FSG [9], MoSS/MoFa [1], gSpan [13], CloseGraph [14], FFSM [6], and Gaston [11]. A related, but slightly different approach is used in Subdue [3].

Most existing work in frequent subgraph mining refers to the case where the input is a database of (attributed) graphs. In this setting it is straightforward to define the support (or frequency) of a subgraph: it is simply the number of graphs in the given database that contain the subgraph. However, if the input is a single large graph (which is the case we consider here), it is surprisingly difficult to find an appropriate support definition, which has certain desirable properties. To the best of our knowledge, there is basically only one suggestion up to now, which is used in the algorithms in [10]: the support of a subgraph is the size of a maximum independent node set of the overlap graph of the embeddings of the subgraph (details of this definition are reviewed in Sections III to V).

However, as we argue in this paper, this approach has the drawback that it can sometimes be too restrictive: there are certain cases in which it allows us to count at most one of two embeddings, even though counting both does not have any harmful effects. In addition, a proper proof was lacking that the support defined in this way is anti-monotone. By studying the core problem of overlapping embeddings in detail, we arrive at a modified support definition and also provide a clear proof that both types of subgraph support are anti-monotone.

II. Frequent Subgraph Mining

In order to fix the algorithmic setting, we briefly review the core principles of frequent subgraph mining. The search is usually restricted to connected subgraphs, since this reduces the search space considerably and suffices for most applications. The search grows all possible (connected) subgraphs, starting from a single node and adding an edge and maybe a node (if it is not yet in the subgraph) in each step. For each grown subgraph the support is computed (for example, as the number of database graphs that contain the subgraph) and infrequent subgraph are eliminated (where infrequent means that the support is below a user-specified minimum support).

Since with this basic procedure the same subgraph can be grown in several ways, namely by adding its nodes and edges in different orders, a fundamental problem is how to avoid redundant search. The predominant method for this is so-called *canonical form pruning*, which is based on the definition of a canonical code word of a graph that uniquely identifies it up to automorphisms. Together with a specific way of growing subgraphs (and thus of building code words), a canonical form can be used to check whether a subgraph has already been considered and thus can be pruned from the search tree [2]. The exact way in which the search space (that is, the subgraph (semi-)lattice) is traversed does not matter much. We use a depth-first search, since this has advantages w.r.t. memory consumption, but a breadth-first approach is also feasible.

It should be noted that the above description does not fix a specific definition of the support of a subgraph. However, it exploits a fundamental property, namely that the support is anti-monotone: the fact that infrequent subgraphs are eliminated from the search presupposes that no supergraph of a subgraph can have a higher support than the subgraph itself. Otherwise it would not be possible to prune infrequent subgraphs, because we would run the risk to miss frequent subgraphs. Since support-based pruning is essential for the efficiency of the mining algorithm, it is of vital importance to ensure that the support is defined in such a way that it is anti-monotone.

III. Embeddings

The fundamental concept underlying any type of support definition for subgraphs is that of an *embedding* of the subgraph into the input graph(s). It is formally defined by the notion of a *subgraph isomorphism*, which we consider in the context of *labeled* or *attributed (simple) graphs*.

Definition 1: A *labeled* or *attributed graph* is a triple $G = (V, E, l)$, where V is the set of vertices, $E \subseteq V \times V - \{(v, v) \mid v \in V\}$ is the set of edges, and $l : V \cup E \to L$ is a function that assigns labels from the set L to nodes and edges.

Definition 2: Let $G = (V_G, E_G, l_G)$ and $S = (V_S, E_S, l_S)$ be two labeled graphs. A *subgraph isomorphism* of S to G is an injective function $f : V_S \to V_G$ satisfying $\forall v \in V_S : l_S(v) = l_G(f(v))$ and $\forall (u, v) \in E_S : (f(u), f(v)) \in E_G \wedge l_S((u, v)) = l_G((f(u), f(v)))$.

Every subgraph isomorphism of a graph S to the input graph(s) defines one *embedding* of S (that is, we use *embedding* as a synonym for *subgraph isomorphism*). Note that two different embeddings may refer to the same nodes and edges, simply by mapping the nodes in a permuted way. Generally, they may share a subset of the nodes, i.e., they may overlap.

Definition 3: Let $G = (V_G, E_G, l_G)$ and $S = (V_S, E_S, l_S)$ be two labeled graphs and f_1 and f_2 two subgraph isomorphisms of S to G. Furthermore, let $V_1 = \{v \in V_G \mid \exists u \in V_S : v = f_1(u)\}$ and $V_2 = \{v \in V_G \mid \exists u \in V_S : v = f_2(u)\}$. That is, let V_1 and V_2 be the images (in G) of the nodes of S. f_1 and f_2 are called *overlapping*, written $f_1 \circledcirc f_2$, iff $V_1 \cap V_2 \neq \emptyset$. f_1 and f_2 are called *equivalent*, written $f_1 \circ f_2$, iff $V_1 = V_2$. Finally, f_1 and f_2 are called *identical*, written $f_1 \equiv f_2$, iff $\forall v \in V_S : f_1(v) = f_2(v)$.

Note that two identical subgraph isomorphisms are actually the same subgraph isomorphism and thus define only one embedding. Note also that identical subgraph isomorphisms are necessarily equivalent, which in turn must be overlapping. The inverse inclusions, however, do not hold in general.

IV. Subgraph Support and Overlap Graphs

In terms of the notions defined in the preceding section the support of a graph S in a given database of graphs is simply the number of graphs into which there exists an embedding of S. However, if the input is a single large graph, this definition is not particularly useful: since there either is an embedding of S into the input graph or not, the support of any subgraph is either 1 or 0. Even worse: if the output is restricted to closed subgraphs (which are subgraphs no supergraph of which has the same support), the mining result is always the input graph (or its connected components) and thus entirely useless.

The most intuitive definition of the support of a subgraph in the single graph setting would be to simply count its embeddings into the input graph. However, this definition is not feasible, since the resulting support is not anti-monotone: the support of a supergraph of a subgraph S can exceed the support of S. This is demonstrated by the simple example shown in Figure 1. (In order to keep things simple, only nodes are labeled, while edges are assumed to be unlabeled, or rather seen as all labeled with the same label, so that it can be neglected.) The graph consisting of only one node labeled A has one embedding into the input graph B-A-B and thus support 1. However, the graph B-A-B, obviously a supergraph of the single node graph, has two embeddings (which are equivalent and differ only in the way in which the nodes labeled B are mapped). Hence it has support 2.

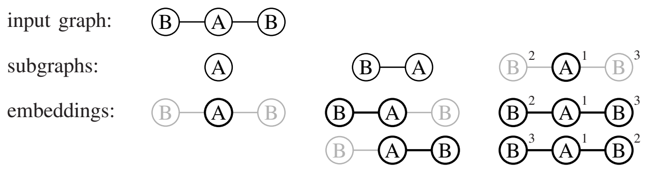

Fig. 1. The number of embeddings of a subgraph is not anti-monotone.

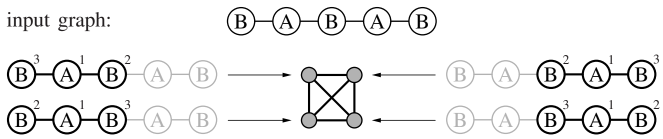

Fig. 2. Overlap graph of the subgraph B-A-B w.r.t. the graph B-A-B-A-B.

The same example also shows that we cannot fix this problem by simply counting only one embedding out of each set of equivalent embeddings (which would be a straightforward and intuitive amendment, since equivalent embeddings are, in a certain sense, also the "same" embedding): the intermediate subgraph A-B has two embeddings—which are not equivalent, because the node labeled B is mapped to different nodes. Hence it also has support 2, violating anti-monotony.

A closer look at this example suggests that the problem is caused by overlapping embeddings (see Definition 3), since the two embeddings of the subgraph A-B overlap on the node labeled A, which, if considered alone, has only one embedding. Therefore it is plausible to consider the overlap graph of the embeddings of a subgraph S and derive the support of S from this overlap graph. This is the basis of the approach used in the algorithms presented in [10].

Definition 4: Let $G = (V_G, E_G, l_G)$ and $S = (V_S, E_S, l_S)$ be two labeled graphs and let V_O be the set of all embeddings (that is, subgraph isomorphisms) of S into G. The *overlap graph* of S w.r.t. G is the graph $O = (V_O, E_O)$, which has the set V_O of embeddings as its node set and the edge set $E_O = \{(f_1, f_2) \mid f_1, f_2 \in V_O \wedge f_1 \not\equiv f_2 \wedge f_1 \circledcirc f_2\}$.

An example of an overlap graph for the subgraph B-A-B w.r.t. the graph B-A-B-A-B is shown in Figure 2. It contains four nodes, one for each of the four possible embeddings. Since in this case every embedding overlaps with every other (because they share at least the node labeled B in the middle of the input graph), the overlap graph is a complete graph.

V. Maximum Independent Set Support

Since overlaps of embeddings appear to be harmful, it is plausible to count at most one of each pair of overlapping embeddings. This leads directly to the idea to define the support of a subgraph as the size of a maximum independent node set of the overlap graph of its embeddings.

Definition 5: Let $G = (V, E)$ be a graph with node set V and edge set $E \subseteq V \times V - \{(v, v) \mid v \in V\}$. An *independent node set* of G is a set $I \subseteq V$ with $\forall u, v \in I : (u, v) \notin E$. I is a *maximum independent node set* iff it is an independent node set and for all independent node sets J of G it is $|I| \geq |J|$.

Note that a maximum independent node set of a graph G need not be unique: there may be several independent node sets having the same (maximum) size. As an example consider again the overlap graph of B-A-B w.r.t. B-A-B-A-B shown in Figure 2. Since this overlap graph is a complete graph, any single node is a maximum independent node set.

Definition 6: Let $O = (V_O, E_O)$ be the overlap graph of the embeddings of a labeled graph $S = (V_S, E_S, l_S)$ into a labeled graph $G = (V_G, E_G, l_G)$. Then the *maximum independent set support* (or *MIS-support* for short) of the graph S w.r.t. the graph G is the size of a maximum independent node set of O.

Since for the example in Figure 2 the overlap graph is complete and thus any maximum independent node set contains at most one node, we conclude that the MIS-support of the graph B-A-B w.r.t. the graph B-A-B-A-B is 1.

The approach of [10] is essentially based on the MIS-support of a graph. The only difference is that [10] defines that two embeddings overlap if they share an edge, while we defined that they overlap if they share a node. Although this leads to minor differences, the basic properties are the same.[1]

However, a fundamental problem of [10] is that no clear and convincing proof is given that MIS-support (with either definition of overlap) is anti-monotone. For this, [10] rather refers to a another paper, namely [12], which, however, turns out to consider a different property (which is difficult to relate to MIS-support) and itself refers to a (hard to obtain) Master's thesis for the details of the proof. Therefore we provide here a clear and fairly straightforward proof, which carries over to our own subgraph support definition, since it only uses properties that are satisfied by our definition as well.

Definition 7: Let $G = (V_G, E_G, l_G)$ and $S = (V_S, E_S, l_S)$ be two labeled graphs and let f be an embedding of S into G. An embedding f' of a proper subgraph $S' = (V', E', l')$ of S (that is, $V' \subset V_S$, $E' = (V' \times V') \cap E_S$, and $l' \equiv l_S|_{V' \cup E'}$) is called an *$S'$-ancestor* of f iff $f' \equiv f|_{V'}$, that is, if the embedding f' coincides with f on the node set V' of S'.

We now consider two very simple observations about S'-ancestors of embeddings, which form the basis of our proof.

Observation 1: For given G, S, S' and f the S'-ancestor f' of the embedding f is uniquely defined.

This is obvious, since f' is only the restriction of f to the node set of S' and thus there is no possibility to choose.

Observation 2: Let $G = (V_G, E_G, l_G)$ and $S = (V_S, E_S, l_S)$ be two labeled graphs and $S' = (V', E', l')$ a proper subgraph of S. Furthermore, let f_1 and f_2 be two (non-identical, but possibly equivalent) embeddings of S into G. f_1 and f_2 overlap if there exist overlapping S'-ancestors f'_1 and f'_2 of the embeddings f_1 and f_2, respectively.

This is obvious, since f'_1 and f'_2 are restrictions of f_1 and f_2 to a subset of the nodes of S. Hence, if f'_1 and f'_2 overlap, so must f_1 and f_2. Note, however, that f_1 and f_2 may overlap even if f'_1 and f'_2 do not overlap: they may overlap on (a subset of) $V - V'$. Hence the reverse implication does not hold.

[1] It has to be noted, though, that defining embeddings as overlapping only if they share an edge makes it possible that they can share several nodes without being considered as overlapping (which may be somewhat unintuitive). It also rules out the possibility to consider single node subgraphs, since such an approach does not rule out violations of anti-monotony occurring in the step from single node subgraphs to subgraphs with two nodes.

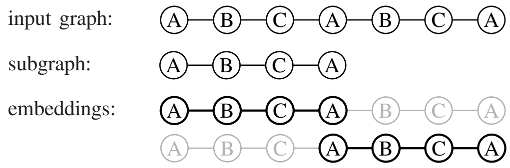

Fig. 3. Not all overlaps of embeddings are harmful.

Theorem 1: MIS-support is anti-monotone.

Proof: We have to show that the MIS-support of a subgraph S w.r.t. a graph G cannot exceed the MIS-support of any (non-empty) proper subgraph S' of S. To do so we consider an arbitrary independent node set I of the overlap O graph of S w.r.t. G. This node set induces a subset I' of the nodes of the overlap graph O' of an (arbitrary, but fixed) subgraph S' of S, which consists of the (uniquely defined, see Observation 1) S'-ancestors of the nodes in I. It is $|I| = |I'|$, because no two elements of I can have the same S'-ancestor: if they did, they would overlap on this ancestor, which would give rise to an edge between them (see Observation 2), contradicting the presupposition that I is an independent node set. With a similar argument we obtain that I' is an independent node set of the overlap graph O': if two nodes of I' were connected by an edge in O', the corresponding two nodes in I would have to be connected by an edge (since there is a pair of overlapping ancestors, see Observation 2), again contradicting the presupposition that I is an independent node set. As a consequence, since I is arbitrary, every independent node set of O induces an independent node set of O' of the same size. Hence the maximum independent node set of O' must be at least as large as the maximum independent node set of O. $\square$

VI. Harmful Overlap Support

The MIS-support definition considers any overlap of two embeddings as harmful: at most one of two overlapping embeddings may be counted. However, there are clearly pairs of embeddings for which the overlap is not harmful, so that there is no reason why they should not both be counted. An example is shown in Figure 3. Even though the two embeddings overlap on the node labeled A in the middle of the input graph, the two embeddings actually do not have an embedding of the subgraph consisting of a single node labeled A as a common ancestor. The reason is that even though there are two nodes labeled A in the subgraph, neither of them is mapped to the same node in both embeddings. Hence the two embeddings cannot be constructed from the same embedding of a single node labeled A, which is then extended in corresponding ways to yield the two embeddings.

In other words, the two embeddings have to be built from two different embeddings of a single node labeled A. Therefore the fact that they overlap does not destroy the anti-monotony of the support. There were two embeddings for any ancestors of these embeddings and thus the support has always been 2. As a consequence, it can be 2 for the subgraph A-B-C-A without harming anti-monotony. This observation gives rise to a more sophisticated support definition.

Definition 8: Let $G = (V_G, E_G, l_G)$ and $S = (V_S, E_S, l_S)$ be two labeled graphs and f_1 and f_2 two subgraph isomorphisms of S to G. f_1 and f_2 are called *harmfully overlapping* (or *H-overlapping* for short), written $f_1 \bullet\!\bullet f_2$, iff they are equivalent (see Definition 3) or there exists a (non-empty) proper subgraph S' of S, so that the S'-ancestors f_1' and f_2' of f_1 and f_2, respectively, are equivalent.

Note that this definition refers to an arbitrary (non-empty) subgraph S' of S. However, the search for frequent subgraphs is usually restricted to connected substructures. As a consequence, only connected ancestors can actually produce counterexamples to anti-monotony, since non-connected subgraphs are not considered in the search. Hence we should restrict the definition to connected subgraphs S' in this case.

Note also that harmful overlap is a weaker concept than simple overlap. The two embeddings in Figure 3 overlap, but they do not overlap harmfully, since they do not possess any equivalent ancestor embeddings. Furthermore, not all overlaps of the embeddings of the graph B-A-B into the graph B-A-B-A-B (as shown in Figure 2) are harmful: the overlaps that give rise to the diagonals of the overlap graph are not harmful (see also Figure 4, which also shows ancestor embeddings and their relationships, and is discussed below).

The support of a subgraph can now be defined in direct analogy to maximum independent set support.

Definition 9: Let $G = (V_G, E_G, l_G)$ and $S = (V_S, E_S, l_S)$ be two labeled graphs and let V_H be the set of all embeddings (that is, subgraph isomorphisms) of S into G. The *harmful overlap graph* of S w.r.t. G is the graph $H = (V_H, E_H)$, which has the set V_H of embeddings as its node set and the edge set $E_H = \{(f_1, f_2) \mid f_1, f_2 \in V_H \wedge f_1 \not\equiv f_2 \wedge f_1 \bullet\!\bullet f_2\}$.

Definition 10: Let $H = (V_H, E_H)$ be the harmful overlap graph of the embeddings of a labeled graph $S = (V_S, E_S, l_S)$ into a labeled graph $G = (V_G, E_G, l_G)$. Then the *harmful overlap support* (or *HO-support* for short) of the graph S w.r.t. G is the size of a maximum independent node set of H.

Theorem 2: HO-support is anti-monotone.

Proof: The proof is exactly the same as for MIS-support (see Theorem 1), since Observations 1 and 2 also hold for HO-support and they were all that was needed for the proof. □

An illustration of the relationships between embeddings of a subgraph and their ancestors is shown in Figure 4. The graphs A, B, A-B, and B-A-B are embedded into the input graph B-A-B-A-B shown at the top. The grey circles are the nodes of harmful overlap graphs for the different subgraphs and correspond to the embeddings shown to the left or to the right of them. Arrows connect ancestors to descendant embeddings (for simplicity, transitive ancestor relations, which are implied, are not shown). Note how any two nodes that are connected by an edge in the square-shaped graph at the bottom (the harmful overlap graph for B-A-B) have equivalent ancestors, while those that are not connected do not.

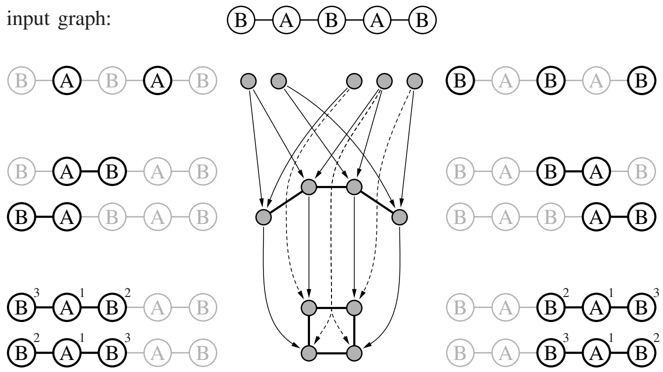

Fig. 4. Harmful overlap graphs of embeddings and ancestor relations.

We conclude that the HO-support of the graph B-A-B w.r.t. the graph B-A-B-A-B is 2, as there are two maximum independent node sets, namely those that consist of the two nodes at the ends of either diagonal of the harmful overlap graph. On the other hand, we found above that the MIS-support is only 1, since a simple overlap graph contains the diagonals (see Figure 2). However, the corresponding overlaps are not harmful (as already pointed out above).

It should be clear that HO-support is never less than MIS-support, since harmful overlap is the weaker concept and thus gives rise to fewer edges in the resulting overlap graph. Therefore, if the same minimum support threshold is used, it cannot be that fewer frequent subgraphs are found with HO-support than with MIS-support. We can rather expect that the number of frequent subgraphs is larger. That the difference can actually be surprisingly large will be seen in Section VIII.

It should also be noted that HO-support is a kind of upper bound for the support of a subgraph if the support is to be anti-monotone, because it only prevents that two embeddings are both counted for the support if they can result from extending equivalent ancestor embeddings (of which at most one was counted). Exceptions are only degenerate cases, where each subgraph giving rise to equivalent ancestors appears again at an independent location in the considered subgraph.

VII. Subgraph Support Computation

It is easy to determine whether two embeddings overlap. Hence computing the overlap graph and thus the MIS-support is fairly simple. (Note, though, that finding a maximum independent node set is NP-complete. What is easy is building the overlap graph.) However, how to build a harmful overlap graph is not quite as simple, since it is more difficult to determine whether two embeddings overlap harmfully. Fortunately, there is still a fairly simple procedure, which is based on trying to construct a subgraph $S_E = (V_E, E_E, l_E)$ that yields equivalent ancestors of two given embeddings f_1 and f_2. The core idea is that for such a subgraph S_E the mapping $g : V_E \to V_E$ with $v \mapsto f_2^{-1}(f_1(v))$, where f_2^{-1} is the inverse of f_2, must be a bijective mapping. More generally, g must be an *automorphism* of S_E, that is, a subgraph isomorphism of S_E to itself. This view of equivalent ancestors leads to the following procedure:

Input: Two (different) embeddings f_1 and f_2 of a labeled graph $S = (V_S, E_S, l_S)$ into a labeled graph $G = (V_G, E_G, l_G)$.
Output: Whether f_1 and f_2 overlap harmfully.

1) Form the sets $V_1 = \{v \in V_G \mid \exists u \in V_S : v = f_1(u)\}$ and $V_2 = \{v \in V_G \mid \exists u \in V_S : v = f_2(u)\}$.
2) Form the sets $W_1 = \{v \in V_S \mid f_1(v) \in V_1 \cap V_2\}$ and $W_2 = \{v \in V_S \mid f_2(v) \in V_1 \cap V_2\}$.
3) If $V_E = W_1 \cap W_2 = \emptyset$, return *false*, otherwise return *true*.

Obviously, V_E, provided it is not empty, is the node set of a subgraph S_E of S that induces equivalent ancestors of f_1 and f_2. On the other hand, any node $v \in V_S - V_E$ cannot contribute to such ancestors, because it is either mapped to a node on which f_1 and f_2 do not overlap (and thus $g(v)$ is not defined) or because there is no original w.r.t. g, that is, no node u with $g(u) = v$ (so that g is not bijective for v). Hence V_E is a maximal set of nodes for which g is a bijection. Therefore: if $V_E \neq \emptyset$, the embeddings f_1 and f_2 overlap harmfully and thus are connected by an edge in the harmful overlap graph. Otherwise any overlap is harmless and we need no edge.

Note that the edge set E_E of the considered subgraph S_E can be found by an analogous construction:

a) Form the sets $E_1 = \{(v_1, v_2) \in E_G \mid \exists (u_1, u_2) \in E_S : (v_1, v_2) = (f_1(u_1), f_1(u_2))\}$ and $E_2 = \{(v_1, v_2) \in E_G \mid \exists (u_1, u_2) \in E_S : (v_1, v_2) = (f_2(u_1), f_2(u_2))\}$.
b) Let $F_1 = \{(v_1, v_2) \in E_S \mid (f_1(v_1), f_1(v_2)) \in E_1 \cap E_2\}$ and $F_2 = \{(v_1, v_2) \in E_S \mid (f_2(v_1), f_2(v_2)) \in E_1 \cap E_2\}$.
c) Let $E_E = F_1 \cap F_2$.

This construction of E_E ensures that the mapping g as defined above is actually an automorphism of the subgraph S_E.

Note, though, that the above procedure assumes that the subgraph S_E always induces equivalent ancestors of two embeddings f_1 and f_2, regardless of its structure. However, the search for frequent subgraphs is usually restricted to connected subgraphs. As a consequence, if the subgraph S_E is not connected, it should not be seen as giving rise to equivalent ancestors, since then S_E does not occur in the search.

Unfortunately, we cannot conclude directly that no edge is needed if the subgraph S_E is not connected, even though an unconnected S_E as a whole does not induce equivalent ancestors: there may be a connected subgraph of S_E that induces equivalent ancestors of the embeddings f_1 and f_2. Hence we have to consider subgraphs of S_E in this case.

For this test we can exploit the following property of the bijective mapping g we defined above (reminder: $\forall v \in V_E : g(v) = f_2^{-1}(f_1(v))$): let $S_C = (V_C, E_C, l_C)$ be an arbitrary (but fixed) connected component of the subgraph S_E and let $W = \{v \in V_C \mid g(v) \in V_C\}$ (that is, let W be the set of nodes of the connected component S_C that are mapped into S_C itself). Then it is either $W = \emptyset$ or $W = V_C$.

We prove this property by contradiction: suppose that there is a connected component S_C for which $W \neq \emptyset$ and $W \neq V_C$. Then we can choose two nodes $v_1 \in W$ and $v_2 \in V_C - W$. These two nodes are connected by a path in S_C, since S_C

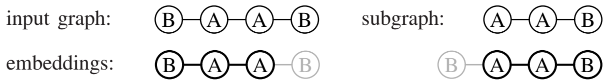

Fig. 5. A simple example of harmful overlap without identical images.

is a connected component. On this path there must be an edge (v_a, v_b) with $v_a \in W$ and $v_b \in V_C - W$. (Along the path there must be (at least) one transition from a node in W to a node not in W, since its start is in W and its end is not in W.) We know that $(v_a, v_b) \in E_E$ (since $E_C \subseteq E_E$) and thus can infer that $(g(v_a), g(v_b)) \in E_E$ (since g is an automorphism of S_E). Since $g(v_a) \in V_C$ (because $v_a \in W$), it follows $g(v_b) \in V_C$, because $g(v_b)$ is connected to $g(v_a)$. However, this implies $v_b \in W$ (due to the construction of W), contradicting $v_b \in V_C - W$, which implies $v_b \notin W$.

With this property the test whether there is a connected subgraph that induces equivalent ancestors becomes very simple: if there is such a subgraph at all, the connected component S_C of S_E that contains it must, as a whole, be such a subgraph as well (otherwise we would have $W \neq \emptyset$ and $W \neq V_C$). Whether a connected component S_C is such a subgraph can be determined by checking for an arbitrary node v of S_C whether $g(v)$ is in S_C or not. If it is, then we know that $W = V_C$ and that g, restricted to the nodes of S_C, is an automorphism of S_C, hence S_C gives rise to equivalent ancestors. If, on the other hand, $g(v)$ is not in S_C, it must be $W = \emptyset$ and neither S_C nor any of its subgraphs induce equivalent ancestors.

The test can be further optimized by the following simple insight: obviously two embeddings f_1 and f_2 overlap harmfully if $\exists v \in V_S : f_1(v) = f_2(v)$, because then such a node v alone gives rise to equivalent ancestors. (There are, of course, also other cases; Figure 5 shows an example. The subgraph inducing equivalent ancestors can be arbitrarily complex even if $\forall v \in V_S : f_1(v) \neq f_2(v)$.) As this test can be performed very quickly, it should be the first step. This has the additional advantage that afterwards we can neglect connected components consisting of isolated nodes, because if such a connected component gives rise to equivalent ancestors, this test would determine it (since the only way in which the mapping g can be an automorphism for an isolated node v is $f_1(v) = f_2(v)$).

As a consequence, we can confine our considerations to the nodes of V_E that are incident to at least one edge in E_E, once we carried out the test for identical images. Therefore the optimized test procedure for harmful overlap is:

Input: Two (different) embeddings f_1 and f_2 of a labeled graph $S = (V_S, E_S, l_S)$ into a labeled graph $G = (V_G, E_G, l_G)$.
Output: Whether f_1 and f_2 overlap harmfully.

1) If $\exists v \in S : f_1(v) = f_2(v)$, return *true*.
2) Form the edge set E_E as described above and the (reduced) node set $V_E' = \{v \in V_S \mid \exists u \in V_S : (v, u) \in E_E\}$.
3) Let $S_C^i = (V_C^i, E_C^i)$, $1 \leq i \leq n$, be the connected components of $S_E' = (V_E', E_E)$. If $\exists i; 1 \leq i \leq n : \exists v \in V_C^i : f_2^{-1}(f_1(v)) \in V_C^i$, return *true*, otherwise return *false*.

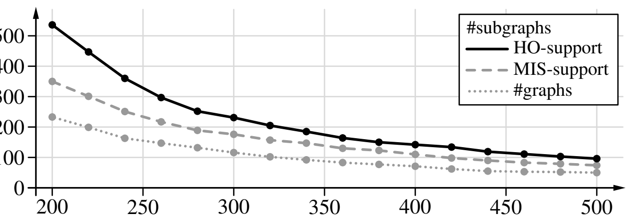

Fig. 6. Experimental results on the IC93 data.

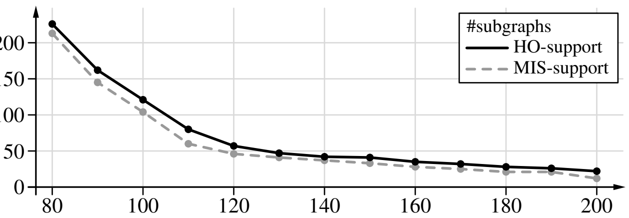

Fig. 7. Experimental results on the Tic-Tac-Toe win data.

VIII. Experimental Results

We implemented the described support computations as part of the MoSS program[2], which is written in Java. This implementation supports an exact computation of a maximum independent node set of an overlap graph as well as a greedy heuristic algorithm, which only very rarely yields results that differ much from the exact results, but is considerably faster.

We applied this implementation to several data sets. In several cases MIS- and HO-support gave the same results, which, however, was mainly due to a lack of overlaps of embeddings. Two examples in which MIS- and HO-support exhibit a different behavior are the IC93 dataset [7], which we interpreted for this application as a single large graph of which each of the 1283 molecules forms a connected component, and the Tic-Tac-Toe win dataset, which is part of the SUBDUE datasets [4] and consists of 626 connected components.

Results on these datasets are shown in Figures 6 and 7, respectively, which depict the number of frequent subgraphs (*not* restricted to closed subgraphs) over absolute minimum support. In both diagrams the solid black line refers to HO-support, the dashed grey line to MIS-support. In Figure 6 we also included the number of subgraphs found with standard support (number of database graphs that contain the subgraph, dotted grey line). While for the Tic-Tac-Toe dataset the increases in the numbers of frequent subgraphs are moderate (around 5%), the difference of HO- and MIS-support on the IC93 dataset is surprisingly large: there are up to 30% more frequent subgraphs. This behavior is mainly due to heavily overlapping molecular fragments with several carbon atoms.

IX. Conclusion

In this paper we studied how to define and compute the support of a subgraph of a single large input graph. We reviewed the MIS-support of [10] and provided a clear and simple proof that it is anti-monotone. In addition, we argued that MIS-support is sometimes too restrictive and suggested an alternative definition (HO-support), which allows embeddings to overlap in certain harmless ways. Only harmful overlaps, which have the effect that the resulting support violates the anti-monotone condition, prevent that both of two overlapping embeddings are counted for the support of a subgraph. Our experiments show that HO-support can sometimes lead to considerably higher support values than MIS-support.

An important point is that HO-support is a kind of upper bound for the support of a subgraph in a single large graph if the support is to be anti-monotone. It only prevents that two embeddings are both counted if they can result from extending equivalent ancestor embeddings (of which at most one was counted). Although there are certain exceptions, it seems to us that it is safe to neglect these degenerate cases.

[2]MoSS is available for download under the Gnu Lesser (Library) Public License at http://www.borgelt.net/moss.html

References

[1] C. Borgelt and M.R. Berthold. Mining Molecular Fragments: Finding Relevant Substructures of Molecules. *Proc. IEEE Int. Conf. on Data Mining (ICDM 2002, Maebashi, Japan)*, 51–58. IEEE Press, Piscataway, NJ, USA 2002

[2] C. Borgelt. On Canonical Forms for Frequent Graph Mining. *Proc. 3rd Int. Workshop on Mining Graphs, Trees and Sequences (MGTS'05, Porto, Portugal)*, 1–12. ECML/PKDD 2005 Organization Committee, Porto, Portugal 2005

[3] D.J. Cook and L.B. Holder. Graph-Based Data Mining. IEEE Trans. on Intelligent Systems 15(2):32–41. IEEE Press, Piscataway, NJ, USA 2000

[4] D.J. Cook and L.B. Holder. SUBDUE datasets. http://cygnus.uta.edu/subdue/download.htm

[5] P.W. Finn, S. Muggleton, D. Page, and A. Srinivasan. Pharmacore Discovery Using the Inductive Logic Programming System PROGOL. *Machine Learning*, 30(2-3):241–270. Kluwer, Amsterdam, Netherlands 1998

[6] J. Huan, W. Wang, and J. Prins. Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism. *Proc. 3rd IEEE Int. Conf. on Data Mining (ICDM 2003, Melbourne, FL)*, 549–552. IEEE Press, Piscataway, NJ, USA 2003

[7] *Index Chemicus — Subset from 1993*. Institute of Scientific Information, Inc. (ISI). Thomson Scientific, Philadelphia, PA, USA 1993 http://www.thomsonscientific.com/products/indexchemicus/

[8] S. Kramer, L. de Raedt, and C. Helma. Molecular Feature Mining in HIV Data. *Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2001, San Francisco, CA)*, 136–143. ACM Press, New York, NY, USA 2001

[9] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. *Proc. 1st IEEE Int. Conf. on Data Mining (ICDM 2001, San Jose, CA)*, 313–320. IEEE Press, Piscataway, NJ, USA 2001

[10] M. Kuramochi and G. Karypis. Finding Frequent Patterns in a Large Sparse Graph. *Proc. 4th SIAM Int. Conf. on Data Mining (SDM 2004, Lake Buena Vista, FL)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA 2004

[11] S. Nijssen and J.N. Kok. A Quickstart in Frequent Structure Mining Can Make a Difference. *Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD2004, Seattle, WA)*, 647–652. ACM Press, New York, NY, USA 2004

[12] N. Vanetik, E. Gudes, and S.E. Shimony. Computing Frequent Graph Patterns from Semistructured Data. *Proc. IEEE Int. Conf. on Data Mining (ICDM 2002, Maebashi, Japan)*, 458–465. IEEE Press, Piscataway, NJ, USA 2002

[13] X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. *Proc. 2nd IEEE Int. Conf. on Data Mining (ICDM 2003, Maebashi, Japan)*, 721–724. IEEE Press, Piscataway, NJ, USA 2002

[14] X. Yan and J. Han. Closegraph: Mining Closed Frequent Graph Patterns. *Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2003, Washington, DC)*, 286–295. ACM Press, New York, NY, USA 2003