cgSpan: Closed Graph-Based Substructure Pattern Mining

Zevin Shaul  
Informatica  
szevin@informatica.com  

Sheikh Naaz  
University of Wisconsin–Madison  
naazish.sheikh@gmail.com  

Abstract—gSpan is a popular algorithm for mining frequent subgraphs. cgSpan (closed graph-based substructure pattern mining) is a gSpan extension that only mines closed subgraphs. A subgraph $g$ is closed in the graphs database if there is no proper frequent supergraph of $g$ that has equivalent occurrence with $g$. cgSpan adds the Early Termination pruning method to the gSpan pruning methods, while leaving the original gSpan steps unchanged.

cgSpan also detects and handles cases in which Early Termination should not be applied. To the best of our knowledge, cgSpan is the first publicly available implementation for closed graphs mining.

Index Terms—frequent graph, graph representation, closed pattern, canonical label

I. INTRODUCTION

The goal of Frequent Subgraph Mining (FSM) is to find subgraphs in a given labeled graphs set that occur more frequently than a given value. This value, known as support, is usually expressed as a percentage of the set size. FSM algorithms can be designed to produce two types of output. The first type outputs all existing frequent subgraphs, while the second type only outputs closed frequent subgraphs. A graph $g$ is closed in a database if there exists no proper frequent supergraph of $g$ that has equivalent occurrence with $g$. The first type of output may have two drawbacks. The first drawback is that the total number of frequent subgraphs discovered becomes very large. For example, if a Star frequent subgraph with $k$ edges is discovered, all $2^k$ subgraphs of a Star graph have the same or a greater support and are therefore also discovered. The second drawback is that not closed frequent subgraphs can be of no interest to the task at hand. For example, frequent parts of the molecule are of no interest in mining chemical graphs set.

gSpan [2] is a popular FSM algorithm that discovers all frequent subgraphs. In this article, we introduce cgSpan, an efficient extension of gSpan that only detects closed frequent graphs. cgSpan was developed to handle the practical use case of ETLs (Extract Transform Load) [3] refactoring. Each ETL can be modeled as a labeled graph. Many ETLs will share common subgraphs that implement the same logic, such as SSN (social security number) field detection and validation. cgSpan allows us to discover such repetitive logic and refactor it into a standalone ETL that is referenced by other ETLs. Such refactoring improves the maintenance, readability, and design of ETLs.

To date there are a number of gSpan implementations that use different programming languages [4] [5] [6]. Such implementations can of course be extended to cgSpan with relatively little programming effort. Our cgSpan implementation, [7], extends the Python implementation [4].

CloseGraph [8] was the first algorithm that was developed for frequent closed subgraphs extraction. cgSpan improves CloseGraph efficiency in two important ways:

(i) cgSpan only examines extensions from the vertices on the right-most path to confirm that frequent subgraph is closed. For the same purpose, CloseGraph must examine extensions from all vertices.

(ii) cgSpan uses an efficient look-up table to check if early termination can be applied to the graph. Only a single lookup of the edge projections set of the last DFS code of the graph is required. After the lookup, the equivalent occurrence is checked only with a very limited number of closed graphs. For the same purpose, CloseGraph must construct all possible extensions of the graph’s parent, check every extension equivalent occurrence with a parent and compare each extension with the graph using the lexicographical order.

Finally, we provide an efficient method to handle early termination failure. We have discovered a number of different cases where applying early termination causes cgSpan to miss closed graphs. Such cases are detected and dealt with.

The rest of the paper is organized as follows. In Section II we provide references to the definitions and notations used in the following sections.

In Sections II-A and II-B we establish the theoretical basis of the cgSpan algorithm. Section II-C formulates the early termination algorithm of cgSpan. Section II-D formulates the method for the detection of early termination failure and the discovery of missing closed frequent subgraphs.

Finally, the cgSpan algorithm is provided in Section II-E. The results of the experiments are reported in Section IV.
II. Preliminary Concepts

The concepts used throughout this paper are listed below. Each concept is accompanied by references to the original definition in [2] and [8].

Definition II.1 (labeled graph). [2] Definition 1, [8] Section 2. A labeled graph has labels associated with its edges and vertices. We denote the vertex set of a graph $g$ by $V(g)$, the edge set by $E(g)$. A function, $l$, can map a vertex or an edge to a label.

Definition II.2 (subgraph isomorphism). [2] Definition 1, [8] Definition 2. A subgraph isomorphism is an injective function $f : V(g) \rightarrow V(g')$, such that (1) $\forall u \in V(g), (l(u) = l'(f(u)))$, and (2) $\forall (u,v) \in E(g), (f(u), f(v)) \in E(g')$ and $l(u,v) = l'(f(u), f((v)))$, where $l$ and $l'$ are the label function of $g$ and $g'$ respectively.

Definition II.3 (occurrence). [8] Definition 5. Let $\varphi(g, g')$ represent the number of possible subgraph isomorphisms of $g$ in $g'$. Given graph $g$ and graph dataset $D = \{G_1, G_2, ..., G_n\}$, the occurrence of $g$ in $D$ is the sum of the number of subgraph isomorphisms of $g$ in every graph of $D$, i.e. $\sum_{i=1}^{n} \varphi(g, G_i)$ denoted by $\mathcal{I}(g, D)$.

Definition II.4 (graph extension). [8] Section 2. A graph $g$ can be extended by adding a new edge $e$. A new graph is denoted by $g \circ_e e$.

Definition II.5 (extendable subgraph isomorphism). [8] Section 4. A graph $g' = g \circ_e e$, $f$ a subgraph isomorphism of $g$ in $G$ and $f'$ a subgraph isomorphism of $g'$ in $G$. If $\exists \rho, \sigma$ a subgraph isomorphism of $g$ in $g'$, $\forall v f(v) = f'(\rho(v))$, then we call $f$ extendable and $f'$ an extended subgraph isomorphism from $f$.

We denote the number of such extendable $f$ by $\phi(g, g', G)$

Definition II.6 (extended occurrence). [8] Definition 6. Given graph $g' = g \circ_e e$ and graph dataset $D = \{G_1, G_2, ..., G_n\}$, the extended occurrence of $g'$ in $D$ w.r.t $g$ is the sum of the number of extendable subgraph isomorphisms of $g$ (w.r.t $g'$) in every graph among $D$, i.e. $\sum_{i=1}^{n} \phi(g, g', G_i)$, denoted by $\mathcal{L}(g, g', D)$.

Definition II.7 (equivalent occurrence). [8] Section 4. Given graph $g' = g \circ_e e$ and graph dataset $D$, if $\mathcal{I}(g, D) = \mathcal{L}(g, g', D)$, we say that $g$ and $g'$ have the equivalent occurrence, which means wherever $g$ occurs in $D$, $g'$ occurs.

Definition II.8 (closed frequent subgraph mining). [8] Section 2. If $g$ is a subgraph of $g'$, then $g'$ is a supergraph of $g$, denoted by $g \subseteq g'$ (proper supergraph, if $g \subset g'$). Given a labeled graph dataset, $D = \{G_1, G_2, ..., G_n\}$, support($g$) (or frequency($g$)) denotes the percentage (or number) of graphs (in $D$) in which $g$ is a subgraph. The set of frequent graph patterns, $FS$, includes all the graphs whose support is no less than a minimum support threshold, $\text{min}_\text{sup}$. The set of closed frequent graph patterns, $CS$, is defined as follows:

$$CS = \{g | g \in FS \text{ and } \exists g' \in FS \text{ such that } g \subset g' \text{ and } g \text{ and } g' \text{ have equivalent occurrence} \}$$

Since $CS$ includes no graph that has a proper supergraph with equivalent occurrence, we have $CS \subseteq FS$. The problem of Closed Frequent Subgraph Mining is to find the complete set of $CS$ in the graph dataset $D$ with a given $\text{min}_\text{sup}$.

Please note that definition of $CS$ in this article is different from $CS$ definition in [8] Section 2. The definition in [8] Section 2 is formulated as $CS = \{g | g \in FS \text{ and } \exists g' \in FS \text{ such that } g \subset g' \text{ and } \text{support}(g) = \text{support}(g')\}$. The reason for this change in definition is that if $\text{support}(g) = \text{support}(g')$, but $\mathcal{I}(g, D) > \mathcal{L}(g, g', D)$, we consider $g$ to be a closed graph in $D$.

For example, in Figure (1a) $g'_1$ is a supergraph of $g'_2$ and both have support = 2. However, $g'_2$ occurs three times in $D$, while $g'_1$ occurs only twice. Therefore, $g'_2$ is considered to be a closed graph.

In the rest of this paper we will simply refer to $CS$ as $S$. 

![Figure 1. Closed frequent graph pattern $CS = \{g'_1, g'_2\}$ of $D = \{G_1, G_2\}$](attachment:fig1.png)

Definition II.9 (DFS Code). [2] Definition 4, [8] Definition 2. Given a DFS tree $T$ for a graph $G$, an edge sequence ($e_i$) can be constructed based on $\prec_{E,T}$, such that
Definition II.10 (DFS Lexicographic Order). Let Z = \{code(G,T) | T is a DFS tree of G\}, i.e., Z is a set containing all DFS codes for all the connected labeled graphs. Suppose there is a linear order \( \prec \) on the set \( E \times L \times L \). DFS Lexicographic Order is a linear order defined as follows. If \( \alpha = code(G_\alpha, T_\alpha) = (a_0, a_1, \ldots, a_m) \) and \( \beta = code(G_\beta, T_\beta) = (b_0, b_1, \ldots, b_n) \), then \( \alpha \prec \beta \) if either of the following is true.

(i) \( \exists t, 0 \leq t \leq \min(m,n), a_k = b_k \text{ for } k < t, a_t \prec b_t \)

(ii) \( a_k = b_k \text{ for } 0 \leq k \leq m, \text{ and } n \geq m \).

Definition II.11 (Minimum DFS Code). Given a graph \( G \), \( Z(G) = \{ code(G, T) | T \text{ is a DFS tree of } G \} \), based on DFS lexicographic order, the minimum one, \( min(Z(G)) \), is called Minimum DFS Code of \( G \). It is also a canonical label of \( G \).

Definition II.12 (DFS Code’s Parent and Child). Given a DFS code \( \alpha = (a_0, a_1, \ldots, a_m) \), any valid DFS code \( \beta = (a_0, a_3, \ldots, a_m, b) \), \( b \) is called \( \alpha \)'s child, and \( \alpha \) is called \( \beta \)'s parent.

Definition II.13 (DFS Code Tree). In a DFS Code Tree, each node represents a DFS code, the relation between parent node and child node complies with the relation described in Definition II.12. The relation between siblings is consistent with the DFS lexicographic order. That is, the pre-order search of DFS Code Tree follows the DFS lexicographic order. The Tree is denoted as \( T \).

Definition II.14 (DFS Code’s Ancestors and Descendants). Given two DFS codes, \( \alpha \) and \( \beta \), in \( T \), if there is a straight path from \( \alpha \) to \( \beta \), then \( \alpha \) is called an ancestor of \( \beta \), and \( \beta \) is called a descendant of \( \alpha \), denoted by \( \text{anc}(\beta) = \{ \text{all ancestors of } \beta \} \), and \( \text{des}(\alpha) = \{ \text{all descendants of } \alpha \} \).

Definition II.15 (right-most extension). Given a graph \( g \) and a DFS tree \( T \) in \( g \), \( e \) can be extended from the right-most vertex connecting to any other vertices on the right-most path (backward extension); or \( e \) can be extended from vertices on the right-most path and introduce a new vertex (forward extension). We call these two kinds of restricted extension as right-most extension: denoted by \( g \diamond_r e \).

III. CGSPAN Algorithm

A. Order of Supergraph Discovery

Theorem III.1. Given two graphs \( G \) and \( G' \), \( G \subset G' \), \( G' \) is a proper supergraph of \( G \), \( \alpha = (a_0, a_1, \ldots, a_m) \) and \( \beta = (b_1, b_2, \ldots, b_n), m > n \) be the DFS codes of \( G \) and \( G' \) respectively when they are discovered for the first time in the DFS Code Tree, then one of the following holds:

(i) \( a_k = b_k \text{ for } 0 \leq k \leq n \text{ i.e. } \alpha \text{ is ancestor of } \beta \)
(ii) \( G' \) is discovered for the first time before \( G \) is discovered for the first time.

Proof. As stated in [2] “According to the definition of Minimum DFS code, the first occurrence of DFS code of a graph in \( T \) (pre-order) is its minimum DFS code.” Therefore \( \alpha = min(\alpha) \) and \( \beta = min(\beta) \).

The proof is by induction on \( n \), the length of \( \alpha \).

Base Case: \( n = 1 \)

If \( a_1 = b_1 \), then (i) is satisfied.

If \( a_1 \neq b_1 \), then \( a_1 > b_1 \). This holds because exists \( b_j \), \( j > 1 \) such that \( a_1 = b_j \). If \( a_1 < b_1 \), we could construct another DFS code of \( G' \) \( \gamma = (b_j, b'_1, \ldots, b'_{m-1}) \). \( \gamma < \beta \), which contradicts \( \beta \) being minimum DFS code.

Since \( a_1 > b_1 \), \( \beta \) is constructed before \( G' \) is discovered before \( G \) and (ii) is satisfied.

Inductive hypothesis: Suppose the theorem holds for all values of \( n \) up to some \( k \), \( k \geq 1 \).

Inductive step: Let \( n = k + 1 \). (i) or (ii) hold for \( 1 \leq n \leq k \) and we need to show that (i) or (ii) hold for \( n = k + 1 \).

Let \( \gamma = \text{min}(\text{parent}(\alpha)) \)

If (i) is true for \( \gamma \), then \( G' \) is discovered before \( \gamma \) (inductive hypothesis), \( \gamma \) is discovered before or at the same time as \( \text{parent}(\alpha) \) (\( \gamma \leq \text{parent}(\alpha) \)) and \( \text{parent}(\alpha) \) is discovered before \( \alpha \). Therefore \( G' \) is discovered before \( \alpha \) and (ii) is true.

If (ii) is true for \( \gamma \), then \( \gamma = (b_1, b_2, \ldots, b_k) \). By definition \( \text{parent}(\alpha) \geq \gamma \).

If \( \text{parent}(\alpha) > \gamma \) then \( \alpha \) is discovered after \( \text{anc}(\gamma) \) (all ancestors of \( \gamma \)). \( \beta \in \text{anc}(\gamma) \) and therefore \( G' \) is discovered before \( \alpha \).

If \( \text{parent}(\alpha) = \gamma \) then \( \alpha = (b_1, b_2, \ldots, b_k, b_j) \), \( j \geq k + 1 \).

If \( j = k + 1 \) then (i) is true.

If \( j > k + 1 \) then \( b_j > b_{k+1} \) and (ii) is true. (If \( b_j > b_{k+1} \) was not true, we could construct another DFS code of \( G' \) \( \delta = (b_1, b_2, \ldots, b_j, b_{k+1}, b'_{k+2}, \ldots, b'_{m}) \). \( \delta < \beta \), which contradicts \( \beta \) being minimum DFS code.)

B. Early Termination Detection

Lemma III.2 and Lemma III.3 provide a theoretical basis for cgSpan early termination detection.

Lemma III.2. For each frequent graph \( g_0 \) in \( D \), exist \( g_1, g_2, \ldots, g_n, n \geq 0 \), such that:

(i) \( g_n \) is a closed graph in \( D \)
(ii) \( g_{i+1} = g_i \circ_e e_i \) i.e. \( g_{i+1} \) is extension of \( g_i \)
(iii) \( g_i \) and \( g_{i+1} \) have equivalent occurrence.

We say that \( g_0 \) and each of \( g_1 \leq i \leq n \) have transitive equivalence occurrence.

Proof. If \( g_0 \) cannot be extended to a graph with equivalent occurrence, then \( g_0 \) is closed by definition and the conditions for \( n = 0 \) are met.

Otherwise \( g_0 \) can be extended to a graph \( g_1, g_1 = g_0 \circ_e e_0 \), so that \( g_0 \) and \( g_1 \) have equivalent occurrence.
By induction, \( g_i \) is either a closed graph or can be extended to a graph \( g_{i+1}, g_{i+1} = g_i \circ_e e_i \), so that \( g_i \) and \( g_{i+1} \) have equivalent occurrence.

Since in each induction step \( i \) the extended graph \( g_i \) is one edge larger than in the previous step, the maximum number of steps \( n \) will not exceed \( \max_{G \in D} |E(G)| \). \( g_n \) cannot be extended to a graph with equivalent occurrence and is therefore closed.

**Lemma III.3.** After the DFS tree search of the graph \( g \) with a DFS code \( (a_1, a_2, \ldots, a_n) \) has been completed, i.e. all graphs whose minimum DFS code starts with \( (a_1, a_2, \ldots, a_n) \) are discovered, all closed graphs that include \( g \), \( \{g'|g' \subseteq g', g' \text{ is closed in } D\} \), are also discovered.

**Proof.** Since every closed graph that contains \( g \) is also a supergraph of \( g \), this lemma follows directly from Theorem III.1.

When a DFS code \( s \) is right-most extended with an edge \( e \), we have to decide whether a further extension of a new graph, \( s \circ_e e \), leads to a closed graph discovery. If this is not the case, the further DFS tree right-most extension of \( s \circ_e e \) should be terminated.

If \( s \circ_e e \) is itself a closed graph in \( D \), we do not terminate its right-most extension.

If \( s \circ_e e \) is not a closed graph in \( D \), cgSpan checks all closed graphs discovered up to this point.

If one of these closed graphs, \( g \), and \( s \circ_e e \) have a transitive equivalence occurrence, there is no need to extend \( s \circ_e e \) any further, and only extensions of \( g \) must be examined to find closed graphs. The DFS tree search of \( g \) was already completed and by Lemma III.3 all closed graphs that include \( g \) have already been discovered. In such a case, cgSpan terminates \( s \circ_e e \) right-most extensions.

If up to this point, no closed graph which has transitive equivalence occurrence with \( s \circ_e e \) has been discovered, from Lemma III.2 we know that such a closed graph exists and from Lemma III.3 we know that such a graph will be discovered when \( s \circ_e e \) will be further-right-most extended. Therefore, \( s \circ_e e \) right-most extension should not be terminated in such a case.

We can conclude that cgSpan never miss an opportunity to terminate DFS tree extension wherever possible.

Let’s see how cgSpan early termination works by examining steps in DFS lexicographical search of \( D = \{G_1, G_2\} \) from Figure 1 as shown in Figure 2. The lexicographical generation order of the generated patterns is: \( g_1, g_2, g_3, g_4 \) and \( g_5 \). When \( g_2 \) is discovered, cgSpan decides whether early termination should be applied to \( g_5 \). This decision is based solely on a fact if \( g_5 \) has a transitive equivalent occurrence with any closed graph discovered so far. The only closed graph discovered before \( g_5 \) was discovered is \( g_4 \). \( I(g_5, D) = L(g_5, g_3, D) = 2 \) and therefore \( g_5 \) and \( g_3 \) have equivalent occurrence. \( I(g_5, D) = L(g_3, g_4, D) = 2 \) and therefore \( g_5 \) and \( g_4 \) have equivalent occurrence. Therefore \( g_5 \) has extended equivalent occurrence with \( g_4 \). Since \( g_5 \) has extended equivalent occurrence with a closed graph \( g_4 \), further right-most extension of \( g_5 \) is early terminated.

**C. Early Termination Implementation**

When a new graph \( s \circ_e e \) is discovered in the DFS search, cgSpan must check whether \( s \circ_e e \) has a transitive equivalent occurrence with any closed graph discovered so far and early terminate \( s \circ_e e \) DFS extension if such a closed graph exists. In fact, we can limit this check to a small number of closed graphs discovered so far by maintaining a closed graphs hash table \( T \).

Assume \( s \circ_e e \) has a transitive equivalent occurrence with a closed graph \( g' \). Let \( F = \{f\} \) and \( F' = \{f'\} \) be sets of isomorphisms of \( s \circ_e e \) and \( g' \) into \( D = \{G_1, G_2, \ldots, G_n\} \) respectively. Then there exists an edge \( e' \in \epsilon' \) such that \( \{f'(e'), f' \in F'\} = \{f(e), f \in F\} \), i.e. \( e' \) and \( e \) are injected into the same set of edges in \( D \).

Therefore \( s \circ_e e \) has to be checked for having transitive equivalent occurrence only with closed graphs with such an edge \( e' \).

Such sets of edges in \( D \) are used as keys in the hash table of the closed graphs. As soon as the closed graph \( g' \) is discovered, we create a hash key \( key_{e'} = \{f'(e'), f' \in F'\} \) for each edge \( e' \in E(g') \) and add entries \( \langle key_{e'}, g' \rangle \) to the hash table of the closed graphs.

The hash table is denoted as \( CGHT \) (Closed Graphs Hash Table).

To make the key hashable, we double index each edge in \( D \) with \((i, j)\) where \( i \) is an index of a graph \( G_i, G_i \in D \) and \( j \) is an edge index in \( G_i \). The double index injective function \( E(G), G \in D \mapsto N \times N \) is denoted as \( EE \) (Edge Enumeration).

Discovered closed graphs are added to the closed graphs hash table using the \texttt{ADD_CLOSED_GRAPH} procedure in Fig. 3. The \texttt{CREATE_EDGE_HASH_KEY} function in Fig. 3 is called to create a hash key.
Table I shows the Edge Enumeration of $D = \{G_1, G_2\}$ in Figure 1 and the hash table of the closed graphs state after the closed graphs $g'_1$ and $g'_2$ of $D$ were discovered.

| Table I | CLOSED GRAPHS HASH TABLE |
|---|---|
| Edge | Enumeration |
| $G_1(v_1, v_2)$ | (1, 1) |
| $G_1(v_1, v_3)$ | (1, 2) |
| $G_1(v_2, v_3)$ | (1, 3) |
| $G_1(v_3, v_4)$ | (1, 4) |
| $G_1(v_1, v_6)$ | (1, 5) |
| $G_2(v_1, v_2)$ | (2, 1) |
| $G_2(v_2, v_3)$ | (2, 2) |
| $G_2(v_2, v_4)$ | (2, 3) |
| $G_2(v_2, v_5)$ | (2, 4) |
| $G_2(v_1, v_6)$ | (2, 5) |

(a) Edge Enumeration of $D = \{G_1, G_2\}$ in Figure 1

To test whether $s \circ e$ has transitive equivalent occurrence with a closed graph $g'$, we must first find all possible isomorphisms $F = \{\rho\}$ of $s \circ e$ into $g'$. To do this, we just have to choose an arbitrary isomorphism $f' \circ g'$ into $G_i \in D$. Next we check all isomorphisms of $s \circ e$ into $G_i \in D$. Every isomorphism $f$ of $s \circ e$ into $G_i \in D$ that satisfies the condition $f(s \circ e) \in F(g')$ defines an isomorphism $\rho$ of $s \circ e$ into $g' \rho(s \circ e) = f^{-1}(f(s \circ e))$. This step is implemented by Lines 6 through 11 of the EARLY_TERMINATION, Fig. 4.

For example, let’s follow variable value assignments by EARLY_TERMINATION, Fig. 4 in the processing of $D = \{G_1, G_2\}$ from Figure 1 when invoked with DFS code $\alpha = \{0, 1, W, a, X, (1, 2, X, d, Z)\}$ and isomorphisms $f_1 : V(\alpha) \rightarrow V(G_1), f_1(0) = v_1, f_1(1) = v_2$ and $f_2 : V(\alpha) \rightarrow V(G_2), f_2(0) = v_1, f_2(1) = v_2, f_2(2) = v_3$. The closed graphs hash table state in this invocation is shown in Table II (b).

To create a key $key_e = \{f(e), f \in F\}$ and test transitive equivalent occurrences only with closed graphs, which are mapped by $key_e$ in the hash table of the closed graphs. This step is implemented by Line 2 and Line 3 of the EARLY_TERMINATION, Fig. 4.
function Early_Termination(s ⊙ r, e, F, CGHT, EE)
  Input:
  s ⊙ r, e - graph checked for early termination
  F - set of isomorphisms of s ⊙ r into edge enumeration of graphs dataset
  CGHT - closed graphs hash table
  EE - edge enumeration of graphs dataset D
  Output:
  true if early termination should be applied to s ⊙ r, e and false otherwise.

  In case of true, also returns g' - the graph for which s ⊙ r, e has transitive equivalent occurrence and ρ - the isomorphisms of s ⊙ r into g'

  hash_key ← Create_Edge_Hash_Key(EE, e, F)
  G' ← CGHT[hash_key]
  for all g' ∈ G' do
    F' ← isomorphisms of g' into D
    P ← ∅
    for all f ∈ F, f : V(s ⊙ r, e) → V(G_i), G_i ∈ D do
      if f(V(s ⊙ r, e)) ⊆ f'(V(g')) then
        create ρ : V(s ⊙ r, e) → V((g'), ρ(v) = f'−1(f(v))
        P ← P ∪ ρ
      end if
    end for
  end for
  if P = ∅ then
goto 4
  end if
  for all ρ ∈ P do
    for all f ∈ F do
      if ∀v ∈ V(s ⊙ r, e) f(v) = f'(ρ(v)) then
        ext_subgraph_isomorphism ← false
      end if
    end for
    if ext_subgraph_isomorphism = false then
goto 17
    end if
  end for
  return true, g', ρ
end function

Figure 4. Early Termination

D. Handling Early Termination Failure

As stated in [8] there are special cases in which early termination cannot be applied. One such example is provided in Figure 5.

cgSpan can effectively handle early termination failure cases. When a new minimum DFS code α = (a_0, a_1, ..., a_m) is constructed by a DFS search, cgSpan checks whether another DFS code β exists so that:

(i) β should not be early terminated
(ii) G_β ⊆ G_α, G_β and G_α are graphs subscripted by DFS codes β and α respectively.
(iii) G_β ⊈ G_{parent(α)}, G_β and G_{parent(α)} are graphs subscripted by DFS codes β and parent(α) = (a_0, a_1, ..., a_{m−1}) respectively. i.e. G_β includes the right-most vertex of α
(iv) β has not yet been discovered

cgSpan does not construct β explicitly, but rather verifies if such β exists by examining each known early termination failure case conditions.

For example, when the DFS code α = [(0, 1, X, a, Y), (1, 2, Y, b, X), (0, 3, X, c, Z)] in DFS search of D = {G_1, G_2} from Figure 5 is discovered, cgSpan detects that the edge Y \(\xrightarrow{b} X\) is breakable according to the definition in [8]. In this case β is the DFS code of the graph created by removing vertex 2 from α.

Such DFS codes, α, which should not be used to terminate other DFS codes are inserted into a separate database using the DETECT_EARLY_TERMINATION_FAILURE procedure in Fig. 6.

The database of DFS codes can be efficiently implemented by a trie like data structure [10] to provide a quick search for the stored DFS codes.

(i) The root node of the trie always represents the null node.
(ii) Each node (except the root) stores a DFS code 5-tuple.
(iii) Child nodes are sorted in lexicographical order.

After the Early Termination conditions in line 29 of EARLY_TERMINATION in Fig. 4 are met, cgSpan applies procedure REJECT_EARLY_TERMINATION in Fig. 6 to check whether an early termination should be rejected.
REJECT_早期终止 find relevant prefixes of the DFS code of terminating closed graph \( g' \). If any of the prefixes exists in Early Termination Failure DFS codes trie storage, early termination is rejected.

For example, for the cgSpan execution on \( D = \{G_1, G_2\} \) from Figure 5 early termination conditions are met for \( s = [(0, 1, X, a, Y), (0, 2, X, c, Z)], closed graph \( CG_1 \) with a DFS code \( \alpha' = [[(0, 1, X, a, Y), (1, 2, Y, b, X), (0, 3, X, c, Z)]] \) and isomorphism \( \rho = \{0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 3\} \) from \( s \) into \( \alpha' \).

As explained above, at this point DFS_Codes_Trie already contains the DFS code \( [(0, 1, X, a, Y), (1, 2, Y, b, X), (0, 3, X, c, Z)] \). In line 10 \( s \) is projected into \( \alpha' \) using \( \rho \). The result is a set of edges in \( \alpha' \) \( A = \{(0, 1, X, a, Y), (0, 3, X, c, Z)\} \).

Line 11 computes the maximum index of edge in \( \alpha' \) that belongs to set \( A \). The edge \( (0, 3, X, c, Z) \) is such an edge and its index in \( \alpha' \) is \( n = 2 \).

The next lines 11 through 15 check whether the \( \alpha' \) prefix \( [[(0, 1, X, a, Y), (1, 2, Y, b, X), (0, 3, X, c, Z)]] \) exists in DFS_Codes_Trie. Since DFS_Codes_Trie contains the prefix \( [[(0, 1, X, a, Y), (1, 2, Y, b, X), (0, 3, X, c, Z)]] \), the early termination is rejected in line 14.

1: procedure DETECT_早期终止失败(\( \alpha, DFS_Codes_Trie \))
   Input:
   \( \alpha \) - DFS code
   DFS_Codes_Trie - Early Termination Failure DFS codes trie storage.
   for all known early termination failure case \( s \) do
   if \( \alpha \) is instance of \( s \) then
   add \( \alpha \) to DFS_Codes_Trie
   end if
   end for
end procedure

8: procedure REJECT_早期终止失败(\( s, g', \rho, DFS_Codes_Trie \))
   Input:
   \( s \) - DFS code
   \( g' \) - closed graph
   \( \rho \) - isomorphism of \( s \) into \( g' \)
   DFS_Codes_Trie - Early Termination Failure DFS codes trie storage.
   \( \alpha' = (a_0, a_1, \ldots, a_n) \) - DFS code of \( g' \)
   \( A = \{a_k | a_k \in \rho(s), 1 \leq k \leq |E(s)|\} \)
   \( n = \max_{a_k \in A} (i_k) \)
   \( \alpha = (a_0, \ldots, a_n) \) - \( \alpha \) is prefix of \( \alpha' \) up to index \( n \)
   if \( \alpha \in DFS_Codes_Trie \) then
   return true
   end if
   return false
end procedure

Figure 6. Early Termination Failure

E. cgSpan Implementation

cgSpan algorithm is provided in Fig. 7.

cgSpan(\( D, min_sup, S \))
   Input: graph dataset \( D \), min_sup.
   Output: The closed frequent graph set \( S \).
   \( 1: S \leftarrow \emptyset \quad \triangleright \) initialize closed frequent graph set
   \( 2: CGHT \leftarrow \emptyset \quad \triangleright \) initialize closed graphs hash table
   \( 3: DFS_Codes_Trie \leftarrow \emptyset \quad \triangleright \) initialize early termination failure DFS codes trie
   \( 4: \) create \( EE \), the Edge Enumeration of \( D \)
   \( 5: S^1 \leftarrow \) all frequent 1-edge graphs in \( D \) together with isomorphisms \( F_e \) of the graph into \( D \)
   \( 6: \) sort \( S^1 \) in DFS lexicographic order
   \( 7: \) for all edge \( e \in S^1 \) do
   \( 8: \) initialize \( s \) with \( e \)
   \( 9: \) SUBGRAPH_MINING(\( s, F_e, min_sup, S, EE, CGHT, DFS_Codes_Trie \))
   end for
   \( 10: \) procedure SUBGRAPH_MINING(\( s, F_e, min_sup, S, EE, CGHT, DFS_Codes_Trie \))
   \( 11: \) if \( s \neq \min(s) \) then
   \( 12: \) return
   \( 13: \) end if
   \( 14: \) \( \) terminate early, \( g, \rho \) \( \triangleright \) EARLY_TERMINATION(\( s, F, CGHT, EE \))
   \( 15: \) if terminate early then
   \( 16: \) if \( \neg \) REJECT_早期终止失败(\( s, g, \rho, DFS_Codes_Trie \)) then
   \( 17: \) return
   \( 18: \) end if
   \( 19: \) end if
   \( 20: \) \( \) DETECT_早期终止失败(\( s, DFS_Codes_Trie \))
   \( 21: \) if \( s \) is frequent in \( D \) then
   \( 22: \) \( C \leftarrow \emptyset \)
   \( 23: \) scan \( D \) once, find every edge \( e \) such that \( s \) can be right-most extended to frequent \( s \circ_e e \)
   \( 24: \) \( F_{s \circ_e e} \leftarrow \) isomorphisms of \( s \circ_e e \) into \( D \)
   \( 25: \) if \( \text{support}(s \circ_e e) \geq \text{min}\_\text{sup} \) then
   \( 26: \) insert \( s \circ_e e \) and \( F_{s \circ_e e} \) into \( C \);
   \( 27: \) end if
   \( 28: \) sort \( C \) in DFS lexicographic order
   \( 29: \) for all \( s \circ_e e \in C \) do
   \( 30: \) SUBGRAPH_MINING(\( s \circ_e e, F_{s \circ_e e}, min\_sup, S, EE, CGHT, DFS_Codes_Trie \))
   \( 31: \) end for
   \( 32: \) if \( C = \emptyset \) or \( \forall s \circ_e e \in C, s \) does not have equivalent occurrence withs \( \circ_e e \) then
   \( 33: \) ADD_CLOSED_GRAPH(\( CGHT, EE, s, F \))
   \( 34: \) insert \( s \) into \( S \);
   \( 35: \) return
   \( 36: \) end if
   \( 37: \) return
   \( 38: \) end procedure

Figure 7. cgSpan algorithm
Step 1 (line 1-3): Initializes data structures.
Step 2 (line 4): Enumerates edges in $D$.
Table II(a) shows an example of such enumeration.
Step 3 (line 5-6): Adds all frequent 1-edge graphs in $D$ and their isomorphisms into $D$ to $S^1$ and sorts them in DFS lexicographic order. After executing this step for $D = \{G_1, G_2\}$ from Figure 1 $S^1$ contains $[(0, 1, W, a, X), (0, 1, W, f, Z), (0, 1, X, b, Y), (0, 1, X, d, Z)]$ with their respective isomorphisms into $D$.
Step 4 (line 12-14): As in gSpan, this step prunes non minimum DFS codes.
Step 5 (line 15-18): This step first checks whether the conditions for early termination are satisfied. See subsection III-C for details. If the early termination conditions evaluate to true, checks whether early termination can be applied. See subsection III-D for details. If this is the case, the further extension of the DFS code $s$ is terminated.
Step 6 (line 21): Detects whether $s$ can cause an early termination failure. See subsection III-D for details.
Step 7 (line 22-31): As in gSpan, finds all frequent right-most extension of $s$. Recursively calls SubgraphMining for each right-most extension following extensions lexicographical order.
Step 8 (line 32-36): If $s$ has no equivalence occurrence with any of its right-most extensions $s \lor e$, adds closed graph $s$ to the result set $S$ and to the closed graphs hash table $CGHT$. 

Theorem III.4. After executing cgSpan($D, min\_sup, S$), graph $G \in S$ iff $G$ is a closed graph in $D$.

Proof.
Frequent subgraph $G$ with a minimum DFS code $s$ will not be added to $S$ only if line 18 in algorithm 7 is reached or step 32 in algorithm 7 evaluates to false for $s$.

- If
  Suppose that $G$ is a closed subgraph in $D$. It is enough to show that line 18 is never reached by a prefix of $s$ and step 32 evaluates to false for $s$.
  Line 18 can only be reached if a prefix of $G$ DFS minimum code $s$ is early terminated by another closed graph. By definition this would be an early termination failure case. Line 17 guarantees that early termination failure cases do not reach line 18.
  Since $G$ is a closed graph, its DFS Code $s$ has no right-most extensions with equivalent occurrence and step 32 in algorithm 7 evaluates to true.
- Only if
  Let $G$ be a frequent not closed graph in $D$. Since $G$ is not closed, it has an extended equivalent occurrence with a closed graph $G'$.
  According to Theorem III.1, either $G'$ is ancestor of $G$ or $G'$ is discovered before $G$.
  If $G'$ is an ancestor of $G$, then step 32 in Algorithm 7 evaluates to false for $G$ ($G$ has right-most extension with equivalent occurrence) and $G$ is not added to $S$.

Let $F'$ be the set of isomorphisms of $G'$ into $D$ and $F$ be the set of isomorphisms of $G$ into $D$. If $G'$ is discovered before $G$, line 33 will add entries (Create\_Edge\_Hash\_Key($EE, e, F'$), $G'$) to $CGHT$ for every edge $e \in E(G')$ before $G$ is discovered. Let $e_s$ be the last edge in $s$. Let $e'$ be $e_s$ matching edge in $G'$. The entry (Create\_Edge\_Hash\_Key($EE, e', F'$), $G'$) $\in$ $CGHT$ when $e_s$ is discovered for $G$. Since $G'$ is an extended equivalent occurrence of $G$, the keys Create\_Edge\_Hash\_Key($EE, e, F'$) and Create\_Edge\_Hash\_Key($EE, e_s, F$) are identical. Therefore, the call to SubgraphMining with $s$ is guaranteed to reach line 18 and $G$ is not added to $S$.

IV. EXPERIMENTS AND PERFORMANCE STUDY
In our experiments we use the two most famous datasets in subgraph mining, Chemical\_340 and Compounds\_422. Both datasets were obtained from the datasets database 11 of the SPMF open source library 12.

The basic characteristics of the Chemical\_340 and Compounds\_422 datasets are summarized in Table II.

| Dataset Name     | Graph count | Average node count per graph | Average edge count per graph | Vertex label count | Edge label count |
|------------------|-------------|------------------------------|------------------------------|-------------------|-----------------|
| Chemical\_340    | 540         | 27.02                        | 27.40                        | 66                | 4               |
| Compounds\_422   | 422         | 39.61                        | 42.31                        | 4                 | 21              |

All experiments are done on a Intel(R) Core(TM) i7-7820HQ CPU @ 2.90GHz PC with 64.0 GB RAM, running 64-bit Windows 10.

cgSpan code is implemented with Python 3.6 and executed with PyPy 7.3.5.

cgSpan code is publicly available in 7.

As part of our experiments, we have validated the completeness of early termination failure handling. The validation process was carried out by executing gSpan and filtering out all not closed frequent graphs from the gSpan output. The closed graphs set obtained by cgSpan execution was validated to be identical to the one obtained by gSpan execution and non closed graphs filtering.

Figure 8 and Figure 9 graphically depict the results of our tests of gSpan and cgSpan on Compounds\_422 and Chemical\_340. The data used to build Figure 8 and Figure 9 can be found in table III and table IV respectively.

In our experiments we found that the output of cgSpan can be roughly 10 percent the size of the output of gSpan, and that the runtime of cgSpan can also be a fraction of the runtime of gSpan. The plots of cgSpan vs gSpan and frequent graphs vs closed frequent graphs in Figure 8 should make our finding readily visible.

In other datasets cgSpan will continue to have a smaller output than gSpan; however, cgSpan may have a longer
The speed at which cgSpan completes compared to gSpan depends on the ratio of closed frequent graphs to frequent graphs in the provided dataset. For the Compounds_422 dataset the ratio is low, as can be seen in Table III column 4, so cgSpan finishes much faster than gSpan. For the Chemical_340 dataset the ratio is higher, see Table IV column 4, so cgSpan is slightly slower than gSpan. The aforementioned phenomena can be seen graphically in Figure 9.

Following our cgSpan vs gSpan testing we conducted further experiments on the value of early termination failure in the cgSpan algorithm. The results of experiments are in Table V and Table VI. We found that depending on the structure of the graphs in a given dataset early termination failure can be vitally important or inconsequential. For the Compounds_422 dataset cgSpan with early termination failure handling can help detect almost 20 percent more graphs than cgSpan without early termination failure handling. For the Chemical_340 dataset cgSpan with early termination failure handling found almost the exact same number of closed frequent graphs as did cgSpan without early termination failure handling. Early termination failure handling is very valuable to the cgSpan algorithm as it helps guarantee the correctness of the algorithm.

cgSpan vs CloseGraph [8] effectiveness can be concluded from the fact that cgSpan outperforms gSpan by a factor of 100 on Compounds_422 dataset when min_sup is close to 5%, while CloseGraph does the same only with a factor of 10.

V. CONCLUSIONS

We have shown that the gSpan algorithm can be efficiently extended to output only closed graphs.

For future work we consider the extension of cgSpan to handle directed graphs. In [13] the extension of gSpan to directed graphs is described. Since cgSpan is an extension of gSpan, the same approach can be used to extend cgSpan to...
directed graphs.

**ACKNOWLEDGMENT**

We would like to express our gratitude to Prof. Doan AnHai for bringing us together to work on this project and providing support and guidance.

**REFERENCES**

[1] Eric W. Weisstein. Star graph. From MathWorld—A Wolfram Web Resource. https://mathworld.wolfram.com/StarGraph.html.

[2] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In 2002 IEEE International Conference on Data Mining, 2002. Proceedings., pages 721–724. IEEE, 2002.

[3] Informatica. What is extract transform load (etl)? https://www.informatica.com/services-and-training/glossary-of-terms/extract-transform-load-definition.html.

[4] Chen Qingying and Karpov Nikolay. gspan. https://github.com/betterenvi/gSpan, 2016.

[5] Nowozin Sebastian and Kudo Taku. gboost. https://github.com/rkwitt/gboost, 2007.

[6] Tony Zhu. gspan.java. https://github.com/TonyZZX/gSpan.Java, 2018.

[7] Zevin Shaul and Sheikh Naa. cgspan. https://github.com/NaaZ03/cgspan, 2021.

[8] Xifeng Yan and Jiawei Han. Closegraph: Mining closed frequent graph patterns. In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’03, page 286–295, New York, NY, USA, 2003. Association for Computing Machinery.

[9] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms, page 253–280. Massachusetts Institute of Technology., 3 edition, 2009.

[10] Edward Fredkin. Trie memory. Commun. ACM, 3(9):490–499, September 1960.

[11] Philippe Fournier-Viger. SPMF Datasets. http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php, 2016.

[12] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Zhihong Deng, and Hoang Thanh Lam. The SPMF open-source data mining library version 2. In Joint European conference on machine learning and knowledge discovery in databases, pages 36–40. Springer, 2016.

[13] Cane Wing-ki Leung. Technical notes on extending gspan to directed graphs. Technical report, Technical Report, Management University, Singapore, 2010.