A Parameterized Method to Solve Graphset-Subgraph Matching Problems with Cycle-Broken Graph Automata

Akio Fujiyoshi
Department of Computer and Information Sciences, Ibaraki University
akio.fujiyoshi.cs@vc.ibaraki.ac.jp

1 Introduction

Let us consider graphset-subgraph matching problems, that is, a type of problems to find a subgraph of an input graph isomorphic to a member of a fixed set of graphs. Here, an input graph may be weighted on both vertices and edges, and we can ask to find a subgraph with the minimum total weight or the maximum total weight. And furthermore, the range of a subgraph may be restricted to spanning, induced, connected or disconnected. Thus various graph problems including some NP-hard problems can be seem as a member of the problems, for example:

Traveling salesman: Given the set of all cycles, to find a connected, spanning subgraph with the minimum total edge weight.

Longest path (cycle): Given the set of all paths (cycles), to find a connected subgraph with the maximum number of edges.

Feedback vertex set: Given the set of all forests, to find a disconnected, induced subgraph with the maximum number of vertices. The set of vertices that are not in the obtained subgraph is a feedback vertex set.

Planarity testing: Given the set of all subdivisions of K_5 or $K_{3,3}$, to find a connected subgraph. If any isomorphic subgraph is not found, then the input graph is planer.

Steiner tree: Given the set of all trees, to find a connected subgraph with the minimum total weight, where an edge-weighted connected graph with a set of terminals is reconstructed to be a connected graph with weighted vertices and weighted edges such that the weight of each edge is the same, the weight of each terminals is -1 times the sum of the weight of the edges, and the weight of other vertices is zero.

Since the sets of graphs listed above can be defined by monadic second-order logic (MSOL), the fixed-parameter tractability of these problems is clear from a theorem of Courcelle [1, 2]. However, a naive implementation of the algorithm obtained from Courcelle's theorem requires the construction of a deterministic finite-state automaton translated from an MSOL formula and the size of the automaton is expected to be a tower of exponentials dependent on the formula and the tree-width. On the other hand, this paper presents a practical parameterized method to solve the problems.

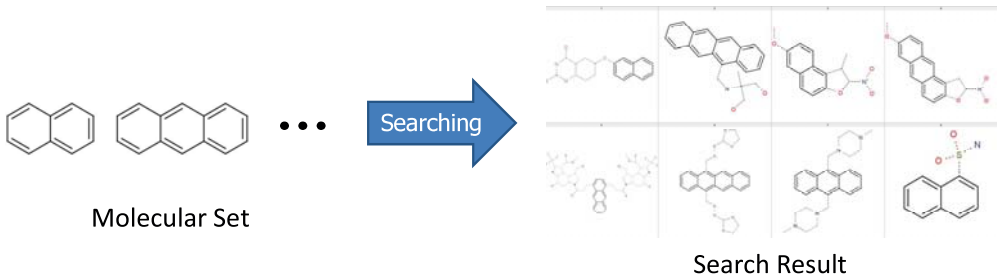

Figure 1: Molecular structure search with given a set of molecules

A practical method for graphset-subgraph matching problems has been requested in the field of chemoinformatics [3, 4]. Pharmaceutical companies and research laboratories have huge databases of chemical structures of drug and drug-candidate molecules. Chemical structures of molecules are labeled, connected graphs, where vertices are labeled as the name of atoms and edges are labeled as the type of bonds. The improvement of molecular structure search technique is a key issue for drug discovery. As the introduction of finite automata and regular expressions for strings [5] revolutionized string search, a similar technique for molecular structure search has been desired. In other words, as shown in Figure 1, given a set of molecules, we want to get a list of subgraph-isomorphic molecules in a database.

A key for the development of a parameterized method for graphset-subgraph matching problems is a way for expressing a fixed set of graphs. The proposed method uses a CBG automaton for defining a set of labeled, connected graphs. CBG automata are a kind of graph automata simply obtained by extending ordinary finite tree automata [6], newly developed for the method. The idea behind this extension is based on a very simple fact: "Any connected graph with cycles becomes a tree if we break all cycles." As shown in Figure 2, by choosing an edge from each cycle and inserting two vertices (broken points) at the middle of the edges, then a tree (a cycle-broken graph) is obtained.

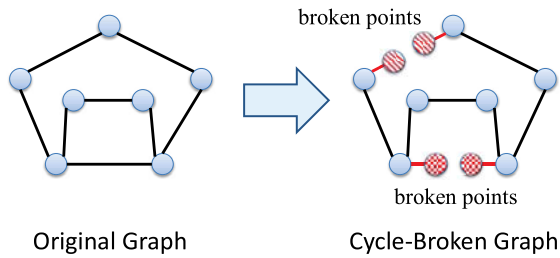

Figure 2: Obtaining a cycle-broken graph

The proposed parameterized method for graphset-subgraph matching problems is developed based on our previous method for finding a spanning tree of an input graph of treewidth 2 isomorphic to a member of a fixed set of trees [7].

2 Preliminaries

A *graph* is an ordered pair $G = (V, E)$, where V is a finite set, called *vertices*, and E is a finite set of unordered pairs of distinct vertices, called *edges*. A vertex $u \in V$ is *adjacent* to another vertex $v \in V$ if an edge $\{u, v\}$ is in E. For $v \in V$, we define $\mathrm{adj}(v) = \{u \mid u \in V$ and u is adjacent to $v\}$, and $|\mathrm{adj}(v)|$ is called the *degree* of v.

For vertices $u, v \in V$, a *path* from u to v is a sequence of vertices $v_1 v_2 \cdots v_n$ for some $n \geq 2$ such that $u = v_1$, $v = v_n$, for $1 \leq i \leq n-1$, v_i is adjacent to v_{i+1}, and $v_1, v_2, \ldots, v_n$ are all distinct except that v_1 may equal to v_n. A *cycle* is a path from v to v for some $v \in V$. G is *acyclic* if there is no cycle in G. G is *connected* if there is a path from u to v for any pair of distinct vertices $u, v \in V$.

A *tree* is a connected, acyclic graph. A vertex of a tree is called a *node*. A *rooted tree* is a pair (T, r) such that T is a tree, and r is a node of T. The node r is called the *root*. In a rooted tree, we assume that the edges have a natural direction away from the root. The *parent* of a node v is the node adjacent to v on the path from the root to v. Note that every node except the root has a unique parent. The *children* of a node v are the nodes whose parent is v. A node without any children is called a *leaf*.

Let Σ be a finite set of *vertex labels*, and let Γ be a finite set of *edge labels*. A *vertex labeling* of G is a function $\sigma : V \to \Sigma$, and a *edge labeling* of G is a function $\gamma : E \to \Gamma$. A *labeled graph* over Σ and Γ is a quadruple $G = (V, E, \sigma, \gamma)$. In this paper, we assume every graph to be labeled, and we use letters in Roman alphabet $A, a, B, b, C, c, \ldots$ for vertex labels and numerical digits $1, 2, 3, \ldots$ for edge labels.

For a connected graph $G = (V, E)$, a *cycle-breaking set* of G is a set of ordered pair of vertices $B \subseteq V \times V$ such that $(v, u) \notin B$ if $(u, v) \in B$, and $G' = (V, E - \{\{u, v\} \mid (u, v) \in B\})$ becomes a tree. In general, there exist a plural number of cycle-breaking sets of G. When G is a tree, however, $B = \emptyset$ is the one and only cycle-breaking set of G.

Let $*$ be a special symbol not included in Σ, and let @ be a special symbol not included in Γ. For a labeled, connected graph $G = (V, E, \sigma, \gamma)$ over Σ and Γ, and a cycle-breaking set B, a *cycle-broken graph* of G decided by B is a labeled, connected graph $G' = (V', E', \sigma', \gamma')$ over $\Sigma \cup \{*\}$ and $\Gamma \cup \{@\}$ defined as follows:

- $V' = V \cup \{u', v' \mid (u, v) \in B\}$, where u' and v' are new vertices not included in V, called *broken points*.
- $E' = E - \{\{u, v\} \mid (u, v) \in B\} + \{\{u, u'\}, \{v, v'\} \mid (u, v) \in B\}$.
- $\sigma' : V' \to \Sigma \cup \{*\}$ is such that, for each $v \in V$, $\sigma'(v) = \sigma(v)$, and, for each $(u, v) \in B$, $\sigma'(u') = \sigma'(v') = *$.
- $\gamma' : E' \to \Gamma \cup \{@\}$ is such that for each $e \in E$, $\gamma'(e) = \gamma(e)$, and, for each $(u, v) \in B$, $\gamma(\{u, u'\}) = @$ and $\gamma(\{v, v'\}) = \gamma(\{u, v\})$.

Note that a cycle-broken graph is always a tree.

Example 1 Consider a labeled, connected graph $G = (V, E, \sigma, \gamma)$ over $\Sigma = \{a, b\}$ and

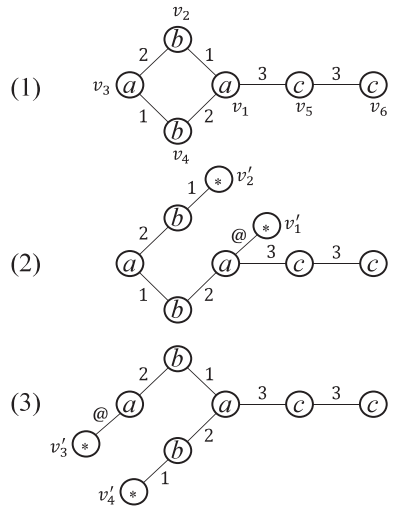

Figure 3: (1) The graph G, (2) the cycle-broken graph decided by B_1, and (3) the cycle-broken graph decided by B_2

$\Delta = \{1, 2, 3\}$, where

$$
\begin{aligned}
V =& \{v_1, v_2, v_3, v_4, v_5, v_6\}, \\
E =& \{\{v_1, v_2\}, \{v_1, v_4\}, \{v_1, v_5\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_5, v_6\}\}, \\
\sigma =& \{(v_1, a), (v_2, b), (v_3, a), (v_4, b), (v_5, b), (v_6, a)\}, \\
& \text{and} \\
\delta =& \{(\{v_1, v_2\}, 1), (\{v_1, v_4\}, 2), (\{v_1, v_5\}, 3), (\{v_2, v_3\}, 2), \\
& (\{v_3, v_4\}, 1), (\{v_5, v_6\}, 3)\}.
\end{aligned}
$$

For the graph G, $B_1 = \{(v_1, v_2)\}$ and $B_2 = \{(v_3, v_4)\}$ are two of the cycle-breaking sets. The graph G and the cycle-broken graphs decided by B_1 and B_2 are illustrated in Figure 3.

3 CBG Automata

For defining a set of labeled, connected graphs, CBG automata were developed [8, 9]. CBG automata are simply obtained by extending ordinary finite tree automata [6] for labeled, ordered trees. A CBG automaton may be said to be a finite tree automaton that accepts cycle-broken graphs instead of labeled, connected graphs.

3.1 Definitions

A *CBG automaton* is a five-tuple $A = (Q, \Sigma, \Gamma, q_0, R)$ where:

- Q is a finite set of *states*,
- Σ is an alphabet of *vertex labels*,

- Γ is an alphabet of *edge labels*,
- $q_0 \in Q$ is the *initial state*, and
- R is a finite set of *transition rules* of the following form:

$$q(f(c_1, c_2, \ldots, c_n)) \to f(q_1(c_1), q_2(c_2), \ldots, q_n(c_n))$$

where $n \geq 0$, $f \in \Sigma$, $q, q_1, q_2, \cdots, q_n \in Q$, and $c_1, c_2, \ldots, c_n \in \Gamma \cup \{@\}$. The number n is called the *width* of a transition rule. When $n = 0$, we write $q(f) \to f$ instead of $q(f()) \to f()$.

Let $A = (Q, \Sigma, \Gamma, q_0, R)$ be a CBG automaton, let $G = (V, E, \sigma, \gamma)$ be a labeled, connected graph, let B be a cycle-breaking set of G, let $G' = (V', E', \sigma', \gamma')$ be the cycle-broken graph of G decided by B, and let $r \in V$ be an vertex of G. A *state mapping* on G' is a function $\mu : V' \to Q$. A state mapping μ on the rooted tree (G', r) is *acceptable* by A if the following conditions hold:

- $\mu(r) = q_0$, i.e., a state mapped to the root is always the initial state,
- for each node $v \in V'$ with n $(n > 0)$ children $v_1, v_2, \ldots, v_n$, if $\sigma(v) = f$, $\mu(v) = q$, $\gamma(\{v, v_1\}) = c_1$, $\gamma(\{v, v_2\}) = c_2$, $\ldots$, $\gamma(\{v, v_n\}) = c_n$, and $\mu(v_1) = q_1$, $\mu(v_2) = q_2$, $\ldots$, $\mu(v_n) = q_n$, then R contains the following transition rule:

$$q(f(c_1, c_2, \ldots, c_n)) \to f(q_1(c_1), q_2(c_2), \ldots, q_n(c_n)),$$

- for each leaf $v \in V'$, if $\sigma(v) = f$, $f \neq *$, and $\mu(v) = q$, then R contains the following transition rule:

$$q(f) \to f,$$

- and for each $(u, v) \in B$, $\mu(u) = \mu(v)$.

G is *accepted* by A if a cycle-breaking set B exists, B decides a cycle-broken graph G', a state mapping μ on G' exists, a vertex $r \in V$ exists, and μ on (G', r) is acceptable by A. The set of connected graphs accpeted by A and the set of (possibly disconnected) graphs accpeted by A are defined as follows:

$$L(A) = \{G \mid G \text{ is accepted by } A\}, \text{ and}$$

$$L_{dc}(A) = \{G \mid \text{each component of } G \text{ is accepted by } A\}.$$

3.2 Examples of CBG Automata

Example 2 $A = (Q, \Sigma, \Gamma, q_0, R)$ is an example of a CBG automaton, where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $\Gamma = \{1, 2, 3\}$, and R consists of transition rules:

$$\begin{aligned}
q_0(a(@, 2, 3)) &\to a(q_1(@), q_2(2), q_3(3)) \\
q_1(a(2)) &\to a(q_2(2)) \\
q_2(b(1)) &\to b(q_1(1)) \\
q_3(b(3)) &\to b(q_3(3)) \\
q_3(a) &\to a
\end{aligned}$$

Consider the graph G and its cycle-breaking set B_1 in Example 1. Let G' be the cycle-broken graph of G decided by B_1. Consider the following state mapping μ on G':

$$\begin{aligned}\mu =&\{(v_1, q_0), (v_2, q_2), (v_3, q_1), (v_4, q_2), (v_5, q_3), (v_6, q_3),\\ &(v'_1, q_1), (v'_2, q_1), \}\end{aligned}$$

The graph G is accepted by A because μ on (G', v_1) is acceptable by A. The state mapping μ on (G', v_1) is illustrated in Figure 4.

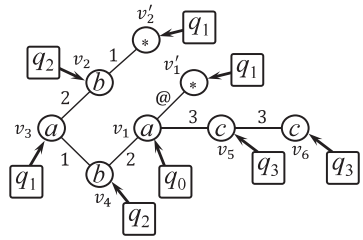

Figure 4: The acceptable state mapping μ on (G', v_1)

Example 3 Another example is CBG Automata that define the set of all paths and the set of all cycles. Since we want to define paths and cycles as non-labeled graphs, we need a CBG automaton for non-labeled graphs. Instead, we set $\Sigma = \{a\}$ and $\Gamma = \{1\}$, i.e., all vertices and all edges have the same labels.

$A = (Q, \Sigma, \Gamma, q_0, R)$ is a CBG automaton that defines the set of all paths, where $Q = \{q_0, q_1\}$, and R consists of transition rules:

$$\begin{aligned} q_0(a(1)) &\rightarrow a(q_1(1)),\\ q_1(a(1)) &\rightarrow a(q_1(1)), \text{ and}\\ q_1(a) &\rightarrow a.\end{aligned}$$

$A = (Q, \Sigma, \Gamma, q_0, R)$ is a CBG automaton that defines the set of all cycles, where $Q = \{q_0, q_1\}$, and R consists of transition rules:

$$\begin{aligned} q_0(a(1, @)) &\rightarrow a(q_1(1), q_1(@)), \text{ and}\\ q_1(a(1)) &\rightarrow a(q_1(1)).\end{aligned}$$

Example 4 The next example is CBG Automata that define all 9 sets of letter graphs. Letter graphs are sets of graphs that represent the topological classification of the capital letters in the Roman alphabet in the Sans Serif font [10]. There are the following 9 sets of letter graphs:

Type A: letter graphs representing AR,

Type B: letter graphs representing B,

Type C: letter graphs representing CIJLMNSUVWZ,

Type D: letter graphs representing DO,

Type E: letter graphs representing EFGTY,

Type H: letter graphs representing HK,

Type P: letter graphs representing P,

Type Q: letter graphs representing Q, and

Type X: letter graphs representing X.

Figure 5 shows the base structures of letter graphs. Each set of letter graphs is defined to be a collection of the subdivisions of the corresponding base structure.

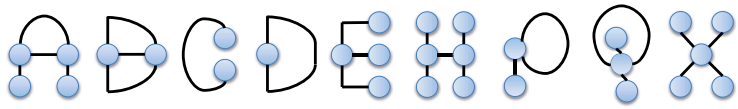

Figure 5: The base structures of letter graphs

Since letter graphs are thought to be non-labeled graphs, we need a CBG automaton for non-labeled graphs. Instead, we set $\Sigma = \{a\}$ and $\Gamma = \{1\}$, i.e., all vertices and all edges have the same labels.

The followings are transition rules of CBG automata that define the 9 sets of letter graphs, where q_0 is the initial state:

- A CBG automaton accepting graphs of type A:

$$\begin{aligned} q_0(a(1,1,@)) &\to a(q_1(1), q_2(1), q_3(@)) \\ q_1(a(1)) &\to a(q_1(1)) \\ q_1(a(1,1)) &\to a(q_3(1), q_2(1)) \\ q_2(a(1)) &\to a(q_2(1)) \\ q_2(a) &\to a \\ q_3(a(1)) &\to a(q_3(1)) \end{aligned}$$

- A CBG automaton accepting graphs of type B:

$$\begin{aligned} q_0(a(1,@,@)) &\to a(q_1(1), q_2(@), q_2(@)) \\ q_1(a(1)) &\to a(q_1(1)) \\ q_1(a(1,1)) &\to a(q_2(1), q_2(1)) \\ q_2(a(1)) &\to a(q_2(1)) \end{aligned}$$

- A CBG automaton accepting graphs of type C:

$$\begin{aligned} q_0(a(1)) &\to a(q_1(1)) \\ q_1(a(1)) &\to a(q_1(1)) \\ q_1(a) &\to a \end{aligned}$$

- A CBG automaton accepting graphs of type D:

$$\begin{aligned} q_0(a(1,@)) &\to a(q_1(1), q_1(@)) \\ q_1(a(1)) &\to a(q_1(1)) \end{aligned}$$

- A CBG automaton accepting graphs of type E:

$$\begin{aligned} q_0(a(1,1,1)) &\to a(q_1(1), q_1(1), q_1(1)) \\ q_1(a(1)) &\to a(q_1(1)) \\ q_1(a) &\to a \end{aligned}$$

- A CBG automaton accepting graphs of type H:

$$\begin{aligned} q_0(a(1,1,1)) &\to a(q_1(1), q_1(1), q_2(1)) \\ q_1(a(1)) &\to a(q_1(1)) \\ q_1(a) &\to a \\ q_2(a(1)) &\to a(q_2(1)) \\ q_2(a(1,1)) &\to a(q_1(1), q_1(1)) \end{aligned}$$

- A CBG automaton accepting graphs of type P:

$$\begin{aligned} q_0(a(1,1,@)) &\to a(q_1(1), q_1(1), q_1(@)) \\ q_1(a(1)) &\to a(q_1(1)) \\ q_1(a) &\to a \end{aligned}$$

- A CBG automaton accepting graphs of type Q:

$$\begin{aligned} q_0(a(1,1,1,@)) &\to a(q_1(1), q_1(1), q_1(1), q_1(@)) \\ q_1(a(1)) &\to a(q_1(1)) \\ q_1(a) &\to a \end{aligned}$$

- A CBG automaton accepting graphs of type X:

$$\begin{aligned} q_0(a(1,1,1,1)) &\to a(q_1(1), q_1(1), q_1(1), q_1(1)) \\ q_1(a(1)) &\to a(q_1(1)) \\ q_1(a) &\to a \end{aligned}$$

3.3 An Extension to CBG automata

A CBG automaton cannot define the set of all trees because the degree of a tree is not bounded. In order to define the set of all trees, we allow a CBG automaton to have the following transition rule:

$$q_0(a(1^*)) \to a(q_0(1)^*)$$

meaning that a vertex to which this rule is applied may have any number of children. The CBG automaton whose transition rules are only the above one defines the set of all trees.

4 A Parameterized Method to Solve the Graphset-Subgraph Matching Problem

In this section, it is shown that graphset-subgraph matching problems with a fixed set of graphs given by a CBG automaton are fixed parameter tractable in the treewidth of an input graph by presenting a parameterized method to solve the problems. Since we have a method to find a spanning tree of an input graph of treewidth 2 isomorphic to a member of a fixed set of trees [7], the proposed method is developed based on the previous method with the following three improvements:

1. A finite tree automaton for a fixed set of trees is replaced by a CBG automaton for a fixed set of graphs.
2. Not only connected, spanning subgraphs but also induced subgraphs, connected subgraphs and disconnected subgraphs become targets to find.
3. The treewidth of an input graph may be any value.

4.1 Outline of the Proposed Method

The purpose of the method is to find an acceptable state mapping on a possible cycle-broken graph of an input graph. A flow of the method is illustrated in Figure 6. Given an elimination ordering of vertices of an input graph, hyperedges representing some part of the graph are combined in succession. First, a hyperedge is arranged for each edge. Each hyperedge is associated with a set of partial solutions. By eliminating a vertex, hyperedges containing the vertex are combined. At the last, there is a hyperedge representing all part of the input graph, and it is associated with a set of solutions (cycle-broken graphs with an acceptable state mapping). One of the most appropriate cycle-broken graphs is supposed to be in the set.

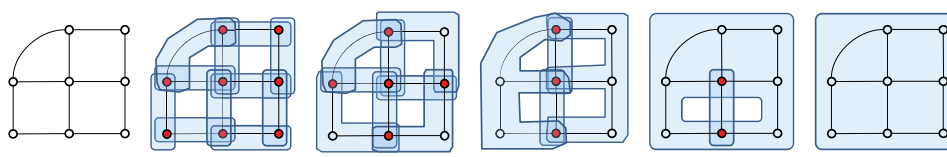

Figure 6: A flow of the method

It is known that there is an elimination ordering of vertices of an input graph so that the size of each hyperedge does not exceed the treewidth of the input graph [11, 12]. A proper elimination ordering can be obtained from a tree decomposition easily.

4.2 Partial Solutions Associated to a Hyperedge

A partial solution is a subgraph of a cycle-broken graph with a partially acceptable state mapping. Because a cycle-broken graph is a rooted tree, each subgraph is a disjoint union of rooted trees. For a hyperedge, only partial solutions that are necessary to obtain a total solution with the minimum or maximum weight should be stored.

Each partial solution associated to a hyperedge has the following representing values: For each vertex in the hyperedge,

- a Boolean value indicating if the vertex is used or not,
- an ID of a transition rule applied to the vertex,
- a selection of IDs of next states in the transition rule which are delivered to the inside of the partial solution, and
- a location (the vertex itself, another vertex in the hyperedge, or one of any already eliminated vertices) of the root of the component to which the vertex belongs. (Possible locations of the root is illustrated in Figure 7.)

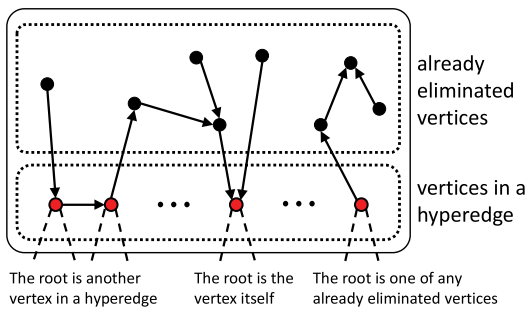

Figure 7: Possible locations of the root

A partial solution is stored as a triple: (1) representing values, (2) a weight, and (3) a set of edges. We store only partial solutions with the minimum or maximum weight among the partial solutions with the same representing values.

4.3 The Initial Setting of Hyperedges

When a hyperedge is arranged for each edge, we associate initial partial solutions to hyperedges. Let $G = (V, E, \sigma, \gamma)$ be a labeled, connected graph for an input, and let $e = \{v_1, v_2\}$ be an edge in E. Let $w : V \cup E \to R$ is a weight function on V and E, where R is the set of real numbers. Suppose that there is a fixed order on V, and $v_1 < v_2$. Suppose also that the following r and r' are transition rules such that $\sigma(v_1) = f$ and $\sigma(v_2) = f'$:

$$r : q(f(c_1, c_2, \ldots, c_n)) \to f(q_1(c_1), q_2(c_2), \ldots, q_n(c_n))$$
$$r' : q'(f'(c'_1, c'_2, \ldots, c'_m)) \to f'(q'_1(c'_1), q'_2(c'_2), \ldots, q'_n(c'_m))$$

For the initial setting, we associate the following partial solutions to the hyperedge $\{v_1, v_2\}$: For all search modes,

- associate $(((true, r, \{i\}, v_1), (true, r', \emptyset, v_1)), w(e), \{e\})$ for some i $(1 \leq i \leq n)$ such that $q' = q_i$ and $c_i = \gamma(e)$, meaning that both v_1 and v_2 are used and v_1 becomes the parent of v_2,
- associate $(((true, r, \emptyset, v_2), (true, r', \{j\}, v_2)), w(e), \{e\})$ for some j $(1 \leq j \leq m)$ such that $q = q'_j$ and $c'_j = \gamma(e)$, meaning that both v_1 and v_2 are used and v_2 becomes the parent of v_1.

- associate $(((true, r, \{i\}, v_1), (true, r', \{j\}, v_2)), w(e), \{e\})$ for some i $(1 \leq i \leq n)$ and j $(1 \leq j \leq m)$ such that $q_i = q'_j$, $c_i = @$ and $c'_j = \gamma(e)$, meaning that both v_1 and v_2 are used and (v_1, v_2) is in a cycle-breaking set, and
- associate $(((true, r, \{i\}, v_1), (true, r', \{j\}, v_2)), w(e), \{e\})$ for some i $(1 \leq i \leq n)$ and j $(1 \leq j \leq m)$ such that $q_i = q'_j$, $c_i = \gamma(e)$ and $c'_j = @$, meaning that both v_1 and v_2 are used and (v_2, v_1) is in a cycle-breaking set.

If search mode is not spanning,

- associate $(((false, -, -, -), (false, -, -, -)), 0, \emptyset)$, meaning that niether v_1 nor v_2 are used,
- associate $(((true, r, \emptyset, v_1), (false, -, -, -)), 0, \emptyset)$, meaning that v_2 is not used, and
- associate $(((false, -, -, -), (true, r', \emptyset, v_2)), 0, \emptyset)$, meaning that v_1 is not used.

And if seach mode is not induced,

- associate $(((true, r, \emptyset, v_1), (true, r', \emptyset, v_2)), 0, \emptyset)$, meaning that both v_1 and v_2 are used but the edge $\{v_1, v_2\} \in E$ is not used.

4.4 Combining Hyperedges

When hyperedges are combined, the partial solutions associated to them are also combined. Because some combinations of partial solutions cause some inconsistency, we need to check (1) the feasibility of transition rules applied to each vertex of the combined hyperedge and (2) the validity of shape of the combined partial solution. Since the shape of a partial solution is a disjoint union of rooted trees, the check points for the validity of shape are the only following two:

1. Each vertex has at most one parent.
2. Each component has exactly one root.

The weight of a combined partial solution is the sum of original partial solutions.

4.5 Eliminating a Vertex in a Hyperedge

When all hyperedges containing a vertex to eliminate are combined to be a single hyperedge, we eliminate the vertex from the hyperedge and the partial solutions associated to it. In order to eliminate a vertex from a partial solution, we need to check if the set of IDs of next states in the transition rule applied to the vertex is fulfilled. In addition, if the vertex is the root of a component, then the state on the left hand-side of the transition rule has to be the initial state.

When a vertex is eliminated from a partial solution, we add the weight of the vertex to the weight of the partial solution.

4.6 Final Condition

When all vertices are eliminated, there is a hyperedge representing all part of the input graph, and the hyperedge is associated with a set of solutions. Each solution is supposed to be a graph accepted by the CBG automaton. We can find a solution with the minimum or maximum weight.

5 Complexity of the Method

Let $G = (V, E, \sigma, \gamma)$ be a labeled, connected graph for an input, and let tw be the treewidth of G. Let $rules$ be the number of transition rules of a CBG automaton, and let $width$ be the maximum width of a transition rule.

The maximum number of partial solutions a hyperedge may have is:

$$T = (|rules| \cdot 2^{width} \cdot tw)^{tw}.$$

Thus the space complexity of the method is $O(|E| \cdot T)$, and the time complexity of the method is $O(|E| \cdot T^2 \cdot \log(T))$.

6 Implementation of the Method

A software for given a labeled, connected graph to find a subgraph accepted by a CBG automaton was developed. The name of the software is CBGfinder. CBGfinder, written in the C++ language, was developed with Visual C++ on Windows. However, it can be compiled by g++ on linux. The source code of CBGfinder is available on the following web site:

`http://apricot.cis.ibaraki.ac.jp/CBGfinder/`.

7 Conclusion

A parameterized method to solve some graphset-subgraph matching problems was proposed. A key for the development of the method was a way for expressing a fixed set of graphs, and the method uses a CBG automaton for that. Because of the versatility of the proposed method, we can say "Any graph problems that can be formalized as a minimization or maximization of a subgraph accepted by a CBG automaton are fixed parameter tractable in the treewidth of an input graph." We have seen that those problems include traveling salesman, longest path, longest cycle, feedback vertex set, planarity testing, and the Steiner tree problem. There are many others.

References

[1] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[2] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

[3] Andrew R. Leach and Valerie J. Gillet. *An Introduction to Chemoinformatics*. Springer, 2007.

[4] Nathan Brown. Chemoinformatics—an introduction for computer scientists. *ACM Comput. Surv.*, 41(2):8:1–8:38, 2009.

[5] Stephen C. Kleene. Representation of events in nerve nets and finite automata. In Claude E. Shannon and John McCarthy, editors, *Automata Studies*, pages 3–42. Princeton University Press, 1951.

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://tata.gforge.inria.fr/`, 2007. release October, 12th 2007.

[7] Akio Fujiyoshi. A practical algorithm for the uniform membership problem of labeled multidigraphs of tree-width 2 for spanning tree automata. *Int. J. Found. Comput. Sci.*, 28(5):563–582, 2017.

[8] 藤芳明生. グラフオートマトンによる部分グラフ探索アルゴリズムの開発と実装. 2016 年度冬の LA シンポジウム, 2017.

[9] 藤芳明生. 閉路分解木オートマトン. 2017 年度夏の LA シンポジウム, 2017.

[10] Rafael López. How does a topologist classify the letters of the alphabet? *ArXiv e-prints*, *https://arxiv.org/abs/1410.3364*, 2014.

[11] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998.

[12] Hans L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *Graph-Theoretic Concepts in Computer Science, 32nd International Workshop, WG 2006, Bergen, Norway, June 22-24, 2006, Revised Papers*, pages 1–14, 2006.