Efficient Algorithms on Trees and Graphs with Unique Node Labels

Gabriel Valiente

Summary. There is a growing interest on trees and graphs with unique node labels in the field of pattern recognition, not only because graph isomorphism and related problems become polynomial-time solvable when restricted to them but also in the light of important practical applications in structural pattern recognition. Current algorithms for testing graph and subgraph isomorphism and computing the graph edit distance, a shortest edit script, a largest common subgraph, and a smallest common supergraph of two graphs with unique node labels, take time quadratic in the number of nodes in the graphs, and the same holds for similar problems on trees with unique node labels. In this paper, simple algorithms are presented for solving these problems in time linear in the number of nodes and edges in the trees or graphs. These new algorithms are based on radix sorting the sets of nodes and edges in the trees or graphs by node label and source and target node label, respectively, followed by a simultaneous traversal of the ordered sets of nodes and edges.

Key words: Graph matching, Trees, Graphs with unique node labels, Graph isomorphism, Subgraph isomorphism, Edit distance, Edit script, Largest common subgraph, Smallest common supergraph, Efficient algorithms

1 Introduction

Graph theoreticians and theoretical computer scientists have been suffering the so-called *graph isomorphism disease* (to establish the complexity of graph isomorphism and related problems) for several decades now [1,2] but, surprisingly, it is the restriction to graphs with unique node labels [3,4] what makes these problems polynomial-time solvable and with important practical applications in pattern recognition [3, 5].

A graph with unique node labels [3] is just a directed graph with nodes labeled over an ordered alphabet such that no two nodes share the same label. Formally, let Σ_V be an ordered set of node labels, and let Σ_E be a set of edge labels. A graph is a four-tuple $G = (V, E, \alpha, \beta)$, where V is a finite set of nodes, $E \subseteq V \times V$ is a finite set of edges, $\alpha : V \to \Sigma_V$ is a node labeling mapping, and $\beta : E \to \Sigma_E$ is an edge labeling mapping. A graph $G = (V, E, \alpha, \beta)$ is a graph with unique node labels if $\alpha(v) \neq \alpha(w)$ for all $v, w \in V$ with $v \neq w$.

G. Valiente: *Efficient Algorithms on Trees and Graphs with Unique Node Labels*, Studies in Computational Intelligence (SCI) **52**, 137–149 (2007)
www.springerlink.com

Graph matching [6] has been studied in the pattern recognition literature in various forms: graph isomorphism [7–10], subgraph isomorphism [9–11] [12, 13], largest common subgraph [14–18], smallest common supergraph [19, 20], and graph edit distance [21–25]. However, a fundamental limitation for the practical application of graph matching in the field of pattern recognition, lies in the complexity of graph matching because subgraph isomorphism, largest common subgraph, smallest common supergraph, and graph edit distance are all NP-complete problems [26].

In the class of graphs with unique node labels, these problems become polynomial-time solvable, because they reduce to the computation of either set union or set intersection for the set of nodes and the set of edges in the graphs. For instance, computing a largest common subgraph of two graphs with unique node labels takes $O(n^2)$ time, where n is the number of nodes [3, 5].

In this paper, we show that the problems of testing graph and subgraph isomorphism and computing the graph edit distance, a shortest edit script, a largest common subgraph, and a smallest common supergraph of two trees or graphs with unique node labels can all be solved in optimal $O(n + m)$ time, where n is the number of nodes and m is the number of edges. The algorithms themselves are not complicated to implement, and they only require the use of standard data structures.

The rest of the paper is organized as follows. In Sect. 2, the notion of graph with unique node labels is recalled. Efficient algorithms for the problems of graph isomorphism, subgraph isomorphism, graph edit distance, shortest edit script, largest common subgraph, and smallest common supergraph on trees and graphs with unique node labels are presented in detail in Sect. 3. Finally, some conclusions are drawn in Sect. 4.

2 Trees and Graphs with Unique Node Labels

The class of graphs with unique node labels, introduced in [3], is characterized by the requirement of each node label being unique. Graphs with unique node labels find application in those problem domains in which objects are modeled by nodes with some property that can be used to uniquely identify them. Some applications of graphs with unique node labels, discussed in [4], include computer network monitoring (where each client, server, or router in a computer network is represented by a node, and an address uniquely identifies such a node in a computer network) and web document analysis (where each unique term that occurs in a document is represented by a node, and multiple occurrences of the same term are represented by the same node). Further application domains for trees and graphs with unique node labels include biochemical networks (where each biochemical reaction in the metabolic pathway of an organism is represented by a node, and multiple occurrences of the same biochemical reaction in the metabolism of an organism are represented by the same node; see [27]) and taxonomic classifications (where each group of species or species name labels a different node of a taxonomic tree; see [28]) in computational biology.

Definition 1. *Let Σ_V be an ordered set of node labels, and let Σ_E be a set of edge labels. A* graph *is a four-tuple $G = (V, E, \alpha, \beta)$, where V is a finite set of nodes, $E \subseteq V \times V$ is a finite set of edges, $\alpha : V \to \Sigma_V$ is a node labeling mapping, and $\beta : E \to \Sigma_E$ is an edge labeling mapping. A graph $G = (V, E, \alpha, \beta)$ is a* graph with unique node labels *if $\alpha(v) \neq \alpha(w)$ for all $v, w \in V$ with $v \neq w$.*

In the class of graphs with unique node labels, the problems of testing graph and subgraph isomorphism and computing the graph edit distance, a shortest edit script, a largest common subgraph, and a smallest common supergraph of two graphs become polynomial-time solvable, because they reduce to computation of either set union or set intersection for the set of nodes and the set of edges in the graphs. For instance, the algorithm given in [3,5] for computing a largest common subgraph of two graphs with unique node labels, can be stated in pseudocode form as follows.

Algorithm 1. Given two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ with unique node labels, compute a largest common subgraph $G = (V, E, \alpha, \beta)$ of G_1 and G_2.

1. $V := \emptyset$
2. **foreach** node $v_1 \in V_1$
   foreach node $v_2 \in V_2$
   if $\alpha_1(v_1) = \alpha_2(v_2)$ **then**
   $V := V \cup \{v\}$, where v is a new node
   $\alpha(v) := \alpha_1(v_1)$
   endif
   endfor
   endfor
3. $E := \emptyset$
4. **foreach** node $v \in V$
   let v_1 be the node of G_1 with $\alpha_1(v_1) = \alpha(v)$
   let v_2 be the node of G_2 with $\alpha_2(v_2) = \alpha(v)$
   foreach node $w \in V$
   let w_1 be the node of G_1 with $\alpha_1(w_1) = \alpha(w)$
   let w_2 be the node of G_2 with $\alpha_2(w_2) = \alpha(w)$
   if $(v_1, w_1) \in E_1$, $(v_2, w_2) \in E_2$ and $\beta_1(v_1, w_1) = \beta_2(v_2, w_2)$ **then**
   $E := E \cup \{(v, w)\}$
   $\beta(v, w) := \beta_1(v_1, w_1)$
   endif
   endfor
   endfor
5. return G

Computation of a largest common subgraph of two graphs with unique node labels using the previous algorithm takes $O(n^2)$ time, where n is the number of nodes in the graphs. A more efficient algorithm is presented in Sect. 3 that only takes $O(n+m)$ time, where n is the number of nodes and m is the number of edges in the graphs.

3 Efficient Algorithms on Trees and Graphs with Unique Node Labels

The problems of testing graph and subgraph isomorphism and computing the graph edit distance, a shortest edit script, the largest common subgraph, and the smallest common supergraph of two graphs with unique node labels, can be solved in time linear in the number of nodes and edges in the graphs, only if the sets of node labels can be sorted in time linear in the number of nodes and the sets of edge source and target node labels can also be sorted in time linear in the number of nodes and edges in the graphs. The procedure was first sketched in [29].

While sorting takes, in general, quasilinear time, there are at least two particular cases of much interest in pattern recognition for which nodes labels can be sorted in linear time. On the one hand, if node labels are small integers, as in [3], let k be a fixed, but arbitrary, constant. Since n integers in the range $\{1, \ldots, kn\}$ can be sorted in $O(n)$ time, by bucket sorting techniques, it follows that the sets of node labels and the sets of edge source and target node labels can be sorted in time linear in the number of nodes and edges in the graphs.

On the other hand, if node labels are strings, as in [5], let k be again a fixed, but arbitrary, constant. Since n strings of total length at most kn can be sorted in $O(n)$ time, by radix sorting techniques [30], it follows that the sets of node labels and the sets of edge source and target node labels can be sorted in time linear in the total length of the strings. In particular, if node labels are all short strings, of $O(1)$ length each.

All trees and graphs are assumed to be given in adjacency list representation in the rest of the paper.

3.1 Graph Isomorphism

Definition 2. *Two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ are* isomorphic *if there is a bijection $\mu : V_1 \to V_2$ such that, for every node $v_i \in V_1$, $\alpha_1(v_i) = \alpha_2(\mu(v_i)$ and for every pair of nodes $v_1, w_1 \in V_1$, $(v_1, w_1) \in E_1$ if and only if $(\mu(v_1), \mu(w_1)) \in E_2$ and $\beta_1(v_1, w_1) = \beta_2(\mu(v_1), \mu(w_1))$. In such a case, μ is a graph isomorphism of G_1 to G_2.*

The efficient computation of the isomorphism of two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ with unique node labels proceeds as follows. Sort V_1 and V_2 by node label and, during a simultaneous traversal [30] of the ordered sets of nodes, map each node $v_1 \in V_1$ to the only node $v_2 \in V_2$ such that $\alpha_1(v_1) = \alpha_2(v_2)$, that is, set $\mu(v_1) = v_2$. In a similar vein, sort E_1 and E_2 by source node label and target node label and then, during a simultaneous traversal of the ordered sets of edges, for each edge $e_1 \in E_1$ and $e_2 \in E_2$, say $e_1 = (v_1, w_1)$ and $e_2 = (v_2, w_2)$, with $\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) = \alpha_2(w_2)$, check that $\beta_1(e_1) = \beta_2(e_2)$. Then, the node mapping $\mu : V_1 \to V_2$ obtained in the first stage is a graph isomorphism of G_1 to G_2 if and only if all nodes of V_1 were mapped and the latter test was successful for all edges of E_1.

Algorithm 2. Given two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ with unique node labels, compute a graph isomorphism μ of G_1 to G_2, if it exists.

1. sort V_1 and V_2 by node label
2. $isomorph := true$
3. **while** $V_1 \neq \emptyset$ and $V_2 \neq \emptyset$ and *isomorph* **do**
   let v_1 and v_2 be the first element of V_1 and V_2, respectively
   if $\alpha_1(v_1) = \alpha_2(v_2)$ **then**
   $\mu(v_1) := v_2$
   $V_1 := V_1 \setminus \{v_1\}$
   $V_2 := V_2 \setminus \{v_2\}$
   else
   $isomorph := false$
   endif
   endwhile
4. sort E_1 and E_2 by target node label
   sort E_1 and E_2 by source node label
5. **while** $E_1 \neq \emptyset$ and $E_2 \neq \emptyset$ and *isomorph* **do**
   let (v_1, w_1) and (v_2, w_2) be the first element of E_1 and E_2
   if $\mu(v_1) = v_2$ and $\mu(w_1) = w_2$ and $\beta_1(v_1, w_1) = \beta_2(v_2, w_2)$ **then**
   $E_1 := E_1 \setminus \{(v_1, w_1)\}$
   $E_2 := E_2 \setminus \{(v_2, w_2)\}$
   else
   $isomorph := false$
   endif
   endwhile
6. return $(\mu, isomorph)$

3.2 Subgraph Isomorphism

Definition 3. *A* subgraph isomorphism *of a graph* $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ *into a graph* $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ *is an injection* $\mu : V_1 \to V_2$ *such that, for every node* $v_i \in V_1, \alpha_1(v_i) = \alpha_2(\mu(v_i)$ *and for every pair of nodes* $v_1, w_1 \in V_1$ *with* $(v_1, w_1) \in E_1, (\mu(v_1), \mu(w_1)) \in E_2$ *and* $\beta_1(v_1, w_1) = \beta_2(\mu(v_1), \mu(w_1))$. *In such a case,* μ *is a subgraph isomorphism of* G_1 *into* G_2.

The efficient computation of a subgraph isomorphism of a graph $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ with unique node labels into another graph $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ with unique node labels proceeds as follows. Sort V_1 and V_2 by node label and, during a simultaneous traversal [30] of the ordered sets of nodes, map each node $v_1 \in V_1$ to the only node $v_2 \in V_2$ such that $\alpha_1(v_1) = \alpha_2(v_2)$, that is, set $\mu(v_1) = v_2$. In a similar vein, sort E_1 and E_2 by source node label and target node label and then, during a simultaneous traversal of the ordered sets of edges, for each edge $e_1 \in E_1$ and $e_2 \in E_2$, say $e_1 = (v_1, w_1)$ and $e_2 = (v_2, w_2)$, with $\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) = \alpha_2(w_2)$, check that $\beta_1(e_1) = \beta_2(e_2)$. Then, the node mapping $\mu : V_1 \to V_2$ obtained in the first stage is a subgraph isomorphism of G_1 into G_2 if and only if all nodes of V_1 were mapped and the latter test was successful for all edges of E_1.

Algorithm 3. Given two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ with unique node labels, compute a subgraph isomorphism μ of G_1 into G_2, if it exists.

1. sort V_1 and V_2 by node label
2. **while** $V_1 \neq \emptyset$ and $V_2 \neq \emptyset$ **do**
   let v_1 and v_2 be the first element of V_1 and V_2, respectively
   if $\alpha_1(v_1) = \alpha_2(v_2)$ **then**
   $\mu(v_1) := v_2$
   $V_1 := V_1 \setminus \{v_1\}$
   endif
   $V_2 := V_2 \setminus \{v_2\}$
   endwhile
3. sort E_1 and E_2 by target node label
   sort E_1 and E_2 by source node label
4. **while** $E_1 \neq \emptyset$ and $E_2 \neq \emptyset$ **do**
   let (v_1, w_1) and (v_2, w_2) be the first element of E_1 and E_2
   if $\mu(v_1) = v_2$ and $\mu(w_1) = w_2$ and $\beta_1(v_1, w_1) = \beta_2(v_2, w_2)$ **then**
   $E_1 := E_1 \setminus \{(v_1, w_1)\}$
   endif
   $E_2 := E_2 \setminus \{(v_2, w_2)\}$
   endwhile
5. $isomorph := (V_1 = \emptyset$ and $E_1 = \emptyset)$
6. return $(\mu, isomorph)$

3.3 Graph Edit Distance

The edit operations of node and edge deletion, insertion, and substitution allow one to transform any given graph into any other graph. In the class of graphs with unique node labels, edge label substitution are allowed but node label substitutions are forbidden, because they may generate graphs with nonunique node labels [3, 4].

A non-negative cost is assigned to each edit operation, the cost of a sequence of edit operations is given by the sum of the individual cost over all of the edit operations in the sequence, and the edit distance of two graphs is defined as the least cost over all sequences of edit operations that transform one graph into the other.

In practical applications, the cost of an edit operations is equal to 1 except for node substitutions, which have infinite cost. Under this assumption of unit cost, the edit distance coincides with the size of a largest common subgraph [21]. Therefore, under the assumption of unit cost, the algorithm for computing a largest common subgraph of two graphs with unique node labels presented below can also be used to compute the edit distance of two graphs with unique node labels.

3.4 Shortest Edit Script

Definition 4. *An* edit script *of a graph $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ to a graph $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ is a set S of edit operations that, if applied in the right order (essentially, inserting an edge only after having inserted the nodes incident with the*

inserted edge), allow one to transform G_1 into G_2. An edit script S of G_1 to G_2 is shortest *if there is no edit script of G_1 to G_2 of smaller size than S.*

The efficient computation of a shortest edit script of two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ with unique node labels proceeds as follows. Sort V_1 and V_2 by node label and, during a simultaneous traversal [30] of the ordered sets of nodes, for each node $v_1 \in V_1$ such that there is no node $v_2 \in V_2$ with $\alpha_1(v_1) = \alpha_2(v_2)$, output the edit operation "delete node $\alpha_1(v_1)$" and for each node $v_2 \in V_2$ such that there is no node $v_1 \in V_1$ with $\alpha_1(v_1) = \alpha_2(v_2)$, output the edit operation "insert node $\alpha_2(v_2)$."

In a similar vein, sort E_1 and E_2 by source node label and target node label and then, during a simultaneous traversal of the ordered sets of edges, for each edge $e_1 \in E_1$ and $e_2 \in E_2$, say $e_1 = (v_1, w_1)$ and $e_2 = (v_2, w_2)$, with $\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) = \alpha_2(w_2)$, if $\beta_1(e_1) \neq \beta_2(e_2)$, then output the edit operation "substitute edge $\alpha_1(v_1)$ to $\alpha_1(w_1)$ label $\beta_2(v_2, w_2)$." Also, for each edge $e_1 = (v_1, w_1) \in E_1$ such that there is no edge $e_2 = (v_2, w_2) \in E_2$ with $\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) = \alpha_2(w_2)$, output the edit operation "delete edge $\alpha_1(v_1)$ to $\alpha_1(w_1)$" and for each edge $e_2 = (v_2, w_2) \in E_2$ such that there is no edge $e_1 = (v_1, w_1) \in E_1$ with $\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) = \alpha_2(w_2)$, output the edit operation "insert edge $\alpha_2(v_2)$ to $\alpha_2(w_2)$."

3.5 Largest Common Subgraph

Definition 5. *A* common subgraph *of two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ is a graph G such that there exist subgraph isomorphisms of G into G_1 and into G_2. A common subgraph G of G_1 and G_2 is* maximal *if there is no subgraph isomorphism of G into any other common subgraph G' of G_1 and G_2, and it is* largest *if there is no common subgraph G' of G_1 and G_2 of larger size than G.*

The efficient computation of a largest common subgraph of two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ with unique node labels proceeds as follows. Let $G = (V, E, \alpha, \beta)$ be an empty graph and let $\gamma : V_1 \to V$ be an array of nodes indexed by the nodes of G_1. Sort V_1 and V_2 by node label and, during a simultaneous traversal [30] of the ordered sets of nodes, for each node $v_1 \in V_1$ and $v_2 \in V_2$ with $\alpha_1(v_1) = \alpha_2(v_2)$, add a new node v to G with $\alpha(v) = \alpha_1(v_1)$ and set $\gamma(v_1) = v$. In a similar vein, sort E_1 and E_2 by source node label and target node label and then, during a simultaneous traversal of the ordered sets of edges, for each edge $e_1 \in E_1$ and $e_2 \in E_2$, say $e_1 = (v_1, w_1)$ and $e_2 = (v_2, w_2)$, with $\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) = \alpha_2(w_2)$, if $\beta_1(e_1) = \beta_2(e_2)$, then add a new edge $e = (v, w)$ to G with $\beta(e) = \beta_1(e_1)$, where $v = \gamma(v_1)$ and $w = \gamma(w_1)$.

Notice that graph and subgraph isomorphism can also be tested by just comparing the size of a largest common subgraph with the size of the given graphs.

3.6 Smallest Common Supergraph

Definition 6. *A* common supergraph *of two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ is a graph G such that there exist subgraph isomorphisms of G_1*

Algorithm 4. Given two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ with unique node labels, output a shortest edit script of G_1 and G_2.

1. sort V_1 and V_2 by node label
2. **while** $V_1 \neq \emptyset$ and $V_2 \neq \emptyset$ **do**
   let v_1 and v_2 be the first element of V_1 and V_2, respectively
   case $\alpha_1(v_1) < \alpha_2(v_2)$
   output "delete node $\alpha_1(v_1)$"
   $V_1 := V_1 \setminus \{v_1\}$
   case $\alpha_1(v_1) > \alpha_2(v_2)$
   output "insert node $\alpha_2(v_2)$"
   $V_2 := V_2 \setminus \{v_2\}$
   otherwise
   $V_1 := V_1 \setminus \{v_1\}$
   $V_2 := V_2 \setminus \{v_2\}$
   endcase
   endwhile
3. sort E_1 and E_2 by target node label
   sort E_1 and E_2 by source node label
4. **while** $E_1 \neq \emptyset$ and $E_2 \neq \emptyset$ **do**
   let (v_1, w_1) and (v_2, w_2) be the first element of E_1 and E_2
   case $\alpha_1(v_1) < \alpha_2(v_2)$ or $(\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) < \alpha_2(w_2))$
   output "delete edge $\alpha_1(v_1)$ to $\alpha_1(w_1)$"
   $E_1 := E_1 \setminus \{(v_1, w_1)\}$
   case $\alpha_1(v_1) > \alpha_2(v_2)$ or $(\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) > \alpha_2(w_2))$
   output "insert edge $\alpha_2(v_2)$ to $\alpha_2(w_2)$"
   $E_2 := E_2 \setminus \{(v_2, w_2)\}$
   otherwise
   if $\beta_1(v_1, w_1) \neq \beta_2(v_2, w_2)$ **then**
   output "substitute edge $\alpha_1(v_1)$ to $\alpha_1(w_1)$ label $\beta_2(v_2, w_2)$"
   endif
   $E_1 := E_1 \setminus \{(v_1, w_1)\}$
   $E_2 := E_2 \setminus \{(v_2, w_2)\}$
   endcase
   endwhile

and G_2 into G. A common supergraph G of G_1 and G_2 is minimal *if there is no subgraph isomorphism into G of any other common supergraph G' of G_1 and G_2, and it is* smallest *if there is no common supergraph G' of G_1 and G_2 of smaller size than G.*

The efficient computation of a smallest common supergraph of two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ with unique node labels proceeds as follows. Let $G = (V, E, \alpha, \beta)$ be an empty graph, and let $\gamma : V_1 \to V$ be an array of nodes indexed by the nodes of G_1. Sort V_1 and V_2 by node label and, during a simultaneous traversal [30] of the ordered sets of nodes, for each node $v_1 \in V_1$ and $v_2 \in V_2$ with $\alpha_1(v_1) = \alpha_2(v_2)$, add a new node v to G with $\alpha(v) = \alpha_1(v_1)$ and set

Algorithm 5. Given two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ with unique node labels, compute a largest common subgraph $G = (V, E, \alpha, \beta)$ of G_1 and G_2.

1. sort V_1 and V_2 by node label
2. $V := \emptyset$
3. **while** $V_1 \neq \emptyset$ and $V_2 \neq \emptyset$ **do**
   let v_1 and v_2 be the first element of V_1 and V_2, respectively
   case $\alpha_1(v_1) < \alpha_2(v_2)$
   $V_1 := V_1 \setminus \{v_1\}$
   case $\alpha_1(v_1) > \alpha_2(v_2)$
   $V_2 := V_2 \setminus \{v_2\}$
   otherwise
   $V := V \cup \{v\}$, where v is a new node
   $\alpha(v) := \alpha_1(v_1)$
   $\gamma(v_1) := v$
   $V_1 := V_1 \setminus \{v_1\}$
   $V_2 := V_2 \setminus \{v_2\}$
   endcase
   endwhile
4. sort E_1 and E_2 by target node label
   sort E_1 and E_2 by source node label
5. $E := \emptyset$
6. **while** $E_1 \neq \emptyset$ and $E_2 \neq \emptyset$ **do**
   let (v_1, w_1) and (v_2, w_2) be the first element of E_1 and E_2
   case $\alpha_1(v_1) < \alpha_2(v_2)$ or $(\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) < \alpha_2(w_2))$
   $E_1 := E_1 \setminus \{(v_1, w_1)\}$
   case $\alpha_1(v_1) > \alpha_2(v_2)$ or $(\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) > \alpha_2(w_2))$
   $E_2 := E_2 \setminus \{(v_2, w_2)\}$
   otherwise
   if $\beta_1(v_1, w_1) = \beta_2(v_2, w_2)$ **then**
   $E := E \cup \{(\gamma(v_1), \gamma(w_1))\}$
   $\beta(\gamma(v_1), \gamma(w_1)) := \beta_1(v_1, w_1)$
   endif
   $E_1 := E_1 \setminus \{(v_1, w_1)\}$
   $E_2 := E_2 \setminus \{(v_2, w_2)\}$
   endcase
   endwhile
7. return G

$\gamma(v_1) = v$ and $\gamma(v_2) = v$. Also, for each node $v_1 \in V_1$ such that there is no node $v_2 \in V_2$ with $\alpha_1(v_1) = \alpha_2(v_2)$, add a new node v to G with $\alpha(v) = \alpha_1(v_1)$ and set $\gamma(v_1) = v$, and for each node $v_2 \in V_2$ such that there is no node $v_1 \in V_1$ with $\alpha_1(v_1) = \alpha_2(v_2)$, add a new node v to G with $\alpha(v) = \alpha_2(v_2)$ and set $\gamma(v_2) = v$.

In a similar vein, sort E_1 and E_2 by source node label and target node label and then, during a simultaneous traversal of the ordered sets of edges, for each edge $e_1 \in E_1$ and $e_2 \in E_2$, say $e_1 = (v_1, w_1)$ and $e_2 = (v_2, w_2)$, with $\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) = \alpha_2(w_2)$, if $\beta_1(e_1) = \beta_2(e_2)$, then add a new edge $e = (v, w)$ to G

with $\beta(e) = \beta_1(e_1)$, where $v = \gamma(v_1)$ and $w = \gamma(w_1)$. Also, for each edge $e_1 = (v_1, w_1) \in E_1$ such that there is no edge $e_2 = (v_2, w_2) \in E_2$ with $\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) = \alpha_2(w_2)$, add a new edge $e = (v, w)$ to G with $\beta(e) = \beta_1(e_1)$, where $v = \gamma(v_1)$ and $w = \gamma(w_1)$, and for each edge $e_2 = (v_2, w_2) \in E_2$ such that there is no edge $e_1 = (v_1, w_1) \in E_1$ with $\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) = \alpha_2(w_2)$, add a new edge $e = (v, w)$ to G with $\beta(e) = \beta_2(e_2)$, where $v = \gamma(v_1)$ and $w = \gamma(w_1)$.

4 Conclusion

Graph matching encompasses a series of related problems with important practical applications in combinatorial pattern matching, pattern recognition, chemical structure search, computational biology, and other areas of engineering and life sciences. In the class of trees and graphs with unique node labels, these problems become polynomial-time solvable and current algorithms for testing graph and subgraph isomorphism and computing the graph edit distance, a shortest edit script, a largest common subgraph, and a smallest common supergraph of two graphs with unique node labels, take time quadratic in the number of nodes in the graphs, and the same holds for similar problems on trees with unique node labels.

The main contribution of this paper is the development of a simple technique for performing set-theoretical operations on the nodes and edges of two trees or graphs with unique node labels. The technique is based on radix sorting the sets of nodes and edges in the trees or graphs by node label and source and target node label, respectively, followed by a simultaneous traversal of the ordered sets of nodes and edges.

Application of this technique to graph matching resulted in simple algorithms for testing graph and subgraph isomorphism and computing the graph edit distance, a shortest edit script, a largest common subgraph, and a smallest common supergraph of two trees or graphs with unique node labels in time linear in the number of nodes and edges in the trees or graphs. The algorithms themselves, for which detailed pseudocode is given, are not complicated to implement, and they only require the use of standard data structures.

Acknowledgment

The research described in this paper was partially supported by BBSRC grant BB/C004310/1, by the Spanish CICYT, project GRAMMARS (TIN2004-07925-C03-01), and by the Japan Society for the Promotion of Science through Long-term Invitation Fellowship L05511 for visiting JAIST (Japan Advanced Institute of Science and Technology).

Algorithm 6. Given two graphs $G_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, E_2, \alpha_2, \beta_2)$ with unique node labels, compute a smallest common supergraph $G = (V, E, \alpha, \beta)$ of G_1 and G_2.

1. sort V_1 and V_2 by node label
2. $V := \emptyset$
3. **while** $V_1 \neq \emptyset$ and $V_2 \neq \emptyset$ **do**
   let v_1 and v_2 be the first element of V_1 and V_2, respectively
   case $\alpha_1(v_1) < \alpha_2(v_2)$
   $V := V \cup \{v\}$, where v is a new node
   $\alpha(v) := \alpha_1(v_1)$
   $\gamma(v_1) := v$
   $V_1 := V_1 \setminus \{v_1\}$
   case $\alpha_1(v_1) > \alpha_2(v_2)$
   $V := V \cup \{v\}$, where v is a new node
   $\alpha(v) := \alpha_2(v_2)$
   $\gamma(v_2) := v$
   $V_2 := V_2 \setminus \{v_2\}$
   otherwise
   $V := V \cup \{v\}$, where v is a new node
   $\alpha(v) := \alpha_1(v_1)$
   $\gamma(v_1) := v$
   $\gamma(v_2) := v$
   $V_1 := V_1 \setminus \{v_1\}$
   $V_2 := V_2 \setminus \{v_2\}$
   endcase
   endwhile
4. sort E_1 and E_2 by target node label
   sort E_1 and E_2 by source node label
5. $E := \emptyset$
6. **while** $E_1 \neq \emptyset$ and $E_2 \neq \emptyset$ **do**
   let (v_1, w_1) and (v_2, w_2) be the first element of E_1 and E_2
   case $\alpha_1(v_1) < \alpha_2(v_2)$ or $(\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) < \alpha_2(w_2))$
   $E := E \cup \{(\gamma(v_1), \gamma(w_1))\}$
   $\beta(\gamma(v_1), \gamma(w_1)) := \beta_1(v_1, w_1)$
   $E_1 := E_1 \setminus \{(v_1, w_1)\}$
   case $\alpha_1(v_1) > \alpha_2(v_2)$ or $(\alpha_1(v_1) = \alpha_2(v_2)$ and $\alpha_1(w_1) > \alpha_2(w_2))$
   $E := E \cup \{(\gamma(v_2), \gamma(w_2))\}$
   $\beta(\gamma(v_2), \gamma(w_2)) := \beta_1(v_2, w_2)$
   $E_2 := E_2 \setminus \{(v_2, w_2)\}$
   otherwise
   if $\beta_1(v_1, w_1) = \beta_2(v_2, w_2)$ **then**
   $E := E \cup \{(\gamma(v_1), \gamma(w_1))\}$
   $\beta(\gamma(v_1), \gamma(w_1)) := \beta_1(v_1, w_1)$
   endif
   $E_1 := E_1 \setminus \{(v_1, w_1)\}$
   $E_2 := E_2 \setminus \{(v_2, w_2)\}$
   endcase
   endwhile
7. return G

References

1. Gati, G.: Further annotated bibliography on the isomorphism disease. J. Graph Theory **3**(1) (1979) 95–109
2. Read, R.C., Corneil, D.G.: The graph isomorphism disease. J. Graph Theory **1**(4) (1977) 339–363
3. Dickinson, P.J., Bunke, H., Dadej, A., Kraetzl, M.: On graphs with unique node labels. In Hancock, E.R., Vento, M., eds.: Proceedings of Fourth IAPR International Workshop on Graph Based Representations in Pattern Recognition. Volume 2726 of Lecture Notes in Computer Science. Berlin Heidelberg New York, Springer (2003) 13–23
4. Dickinson, P.J., Bunke, H., Dadej, A., Kraetzl, M.: Matching graphs with unique node labels. Pattern Anal. Appl. **7**(3) (2004) 243–254
5. Schenker, A., Bunke, H., Last, M., Kandel, A.: Polynomial time complexity graph distance computation for web content mining. In Basu, M., Ho, T.K., eds.: Data Complexity in Pattern Recognition. Berlin Heidelberg New York, Springer (2006)
6. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. Int. J. Pattern Recogn. Artif. Intell. **18**(3) (2004) 265–298
7. Jiang, X.Y., Bunke, H.: Including geometry in graph representations: A quadratic-time graph isomorphism algorithm and its applications. In: Proceedings of Sixth International Workshop on Structural and Syntactical Pattern Recognition. Volume 1121 of Lecture Notes in Computer Science. Berlin Heidelberg New York, Springer (1996) 110–119
8. Jiang, X.Y., Bunke, H.: Optimal quadratic-time isomorphism of ordered graphs. Pattern Recogn. **32**(7) (1999) 1273–1283
9. Messmer, B.T., Bunke, H.: Error-correcting graph isomorphism using decision trees. Int. J. Pattern Recogn. **12**(6) (1998) 721–742
10. Messmer, B.T., Bunke, H.: A decision tree approach to graph and subgraph isomorphism detection. Pattern Recogn. **32**(12) (1999) 1979–1998
11. Jiang, X.Y., Bunke, H.: Marked subgraph isomorphism of ordered graphs. In: Proceedings of Joint IAPR International Workshops on Structural and Syntactical Pattern Recognition and Structural Pattern Recognition. Volume 1451 of Lecture Notes in Computer Science. Berlin Heidelberg New York, Springer (1998) 122–131
12. Wong, A.K.C., You, M., Chan, S.C.: An algorithm for graph optimal monomorphism. IEEE Trans. Syst. Man Cybern. **20**(3) (1990) 628–636
13. Wong, E.K.: Model matching in robot vision by subgraph isomorphism. Pattern Recogn. **25**(3) (1992) 287–303
14. Akinniyi, F.A., Wong, A.K.C., Stacey, D.A.: A new algorithm for graph monomorphism based on the projections of the product graph. IEEE Trans. Syst. Man Cybern. **16**(5) (1986) 740–751
15. Bunke, H., Messmer, B.T.: Recent advances in graph matching. Int. J. Pattern Recogn. **11**(1) (1997) 169–203
16. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern Recogn. Lett. **19**(3–4) (1998) 255–259
17. Hidovic, D., Pelillo, M.: Metrics for attributed graphs based on the maximal similarity common subgraph. Int. J. Pattern Recogn. Artif. Intell. **18**(3) (2004) 299–313
18. Wallis, W.D., Shoubridge, P., Kraetz, M., Ray, D.: Graph distances using graph union. Pattern Recogn. Lett. **22**(6–7) (2001) 701–704
19. Bunke, H., Jiang, X.Y., Kandel, A.: On the minimum common supergraph of two graphs. Computing **65**(1) (2000) 13–25

20. Fernández, M.L., Valiente, G.: A graph distance measure combining maximum common subgraph and minimum common supergraph. Pattern Recogn. Lett. **22**(6–7) (2001) 753–758
21. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recogn. Lett. **18**(8) (1997) 689–694
22. Bunke, H.: Error-tolerant graph matching: A formal framework and algorithms. In: Advances in Pattern Recognition. Volume 1451 of Lecture Notes in Computer Science. Berlin Heidelberg New York, Springer (1998) 1–14
23. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. Syst. Man Cybern. **13**(3) (1983) 353–363
24. Shapiro, L.G., Haralick, R.M.: Structural descriptions and inexact matching. IEEE Trans. Pattern Anal. **3**(5) (1981) 504–519
25. Tsai, W.H., Fu, K.S.: Error-correcting isomorphism of attributed relational graphs for pattern analysis. IEEE Trans. Syst. Man Cybern. **9**(12) (1979) 757–769
26. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to NP-Completeness. New York, Freeman (1979)
27. Deville, Y., Gilbert, D., van Helden, J., Wodak, S.J.: An overview of data models for the analysis of biochemical pathways. Brief. Bioinform. **4**(3) (2003) 246–259
28. Page, R.D.M., Valiente, G.: An edit script for taxonomic classifications. BMC Bioinform. **6** (2005) 208
29. Valiente, G.: Comment to "Polynomial time complexity graph distance computation for web content mining". In Basu, M., Ho, T.K., eds.: Data Complexity in Pattern Recognition. Berlin Heidelberg New York, Springer (2006)
30. Valiente, G.: Algorithms on Trees and Graphs. Berlin Heidelberg New York, Springer (2002)