Combinatorial algorithms
computing graph isomorphism,
computing tree isomorphism

Jiří Vyskočil, Radek Mařík
2013
Computing Graph Isomorphism

**definition:**

Two graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ are *isomorphic* if there is a bijection $f: V_1 \rightarrow V_2$ such that

$$\forall x, y \in V_1 : \{ f(x), f(y) \} \in E_2 \iff \{ x, y \} \in E_1$$

The mapping $f$ is said to be an *isomorphism* between $G_1$ and $G_2$.

**example:**

\[ G_1 : \quad G_2 : \quad f : \]

\[
\begin{align*}
G_1 & : \\
& a \quad g \\
& b \quad h \\
& c \quad i \\
& d \quad j
\end{align*}
\]

\[
\begin{align*}
G_2 & : \\
& 1 \quad 2 \\
& 5 \quad 6 \\
& 8 \quad 7 \\
& 4 \quad 3
\end{align*}
\]

\[
\begin{align*}
f(a) &= 1 \\
f(b) &= 6 \\
f(c) &= 8 \\
f(d) &= 3 \\
f(g) &= 5 \\
f(h) &= 2 \\
f(i) &= 4 \\
f(j) &= 7
\end{align*}
\]
Computing Graph Isomorphism

Problem:

The graph isomorphism problem is the computational problem of determining whether two finite graphs are isomorphic.

- The graph isomorphism problem is one of a very small number of problems belonging to NP neither known to be solvable in polynomial time nor NP-complete.
- However, there is a number of important special cases of the graph isomorphism problem that have efficient, polynomial-time solutions: trees, planar graphs, some bounded-parameter graphs, etc.
Computing Graph Isomorphism

definition of invariant:

Let $\mathcal{F}$ be a family of graphs. An invariant on $\mathcal{F}$ is a function $\Phi$ with domain $\mathcal{F}$ such that

$$\forall G_1, G_2 \in \mathcal{F} : \Phi(G_1) = \Phi(G_2) \iff G_1 \text{ is isomorphic to } G_2$$

example:

- $|V|$ for graph $G=(V, E)$ is an invariant.
- The following degree sequence $[\deg(v_1), \deg(v_2), \deg(v_3), \ldots, \deg(v_n)]$ is not an invariant.
- However, if the degree sequence is sorted in non-decreasing order, then it is an invariant.
Computing Graph Isomorphism

**definition:**

Let $\mathcal{F}$ be a family of graphs on vertex set $V$ and let $D$ be a function with domain ($\mathcal{F} \times V$). Then the *partition $B_G$ of $V$ induced by $D$* is

$$B_G = [ B_G[0], B_G[1], \ldots, B_G[n-1] ]$$

where

$$B_G[i] = \{ v \in V : D(G, v) = i \}$$

If the function

$$\Phi_D(G) = [ |B_G[0]|, |B_G[1]|, \ldots, |B_G[n-1]| ]$$

is an invariant, then we say that $D$ is an *invariant inducing function.*
Computing Graph Isomorphism - Example

Let

- \( D_1(G, x) = \text{deg}_G(x) \)
- \( D_2(G, x) = \{d_j(x) : j = 1, 2, \ldots, \max\{\text{deg}_G(x) : x \in V(G)\}\} \)

where \( d_j(x) = |\{y : y \text{ is adjacent to } x \text{ and } \text{deg}_G(y) = j\}| \)

Suppose the following graphs \( G_1 \) and \( G_2 \):
Computing Graph Isomorphism - Example

\[ X_0(\mathcal{G}_1) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}. \]

\[ X_0(\mathcal{G}_2) = \{a, b, c, d, e, f, g, h, i, j\}. \]

\[
\begin{array}{c|cccccccccc}
 x & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
 D_1(\mathcal{G}_1, x) & 1 & 3 & 3 & 6 & 3 & 6 & 3 & 3 & 3 & 1 \\
\end{array}
\]

\[ X_1(\mathcal{G}_1) = \{0, 9\}, \{1, 2, 4, 6, 7, 8\}, \{3, 5\} \]

\[
\begin{array}{c|cccccccc}
 x & a & b & c & d & e & f & g & h & i & j \\
\hline
 D_1(\mathcal{G}_2, \bar{x}) & 3 & 3 & 3 & 3 & 6 & 6 & 3 & 3 & 1 & 1 \\
\end{array}
\]

\[ X_1(\mathcal{G}_2) = \{i, j\}, \{a, b, c, d, g, h\}, \{e, f\}. \]
Computing Graph Isomorphism - Example

\[ D_2(G_1, 0) = (0, 0, 1, 0, 0, 0, 0, 0, 0, 0) \]
\[ D_2(G_1, 1) = (0, 0, 2, 0, 0, 1, 0, 0, 0) \]
\[ D_2(G_1, 2) = (0, 0, 1, 0, 0, 2, 0, 0, 0) \]
\[ D_2(G_1, 3) = (0, 0, 5, 0, 0, 1, 0, 0, 0) \]
\[ D_2(G_1, 4) = (0, 0, 1, 0, 0, 2, 0, 0, 0) \]
\[ D_2(G_1, 5) = (0, 0, 5, 0, 0, 1, 0, 0, 0) \]
\[ D_2(G_1, 6) = (0, 0, 1, 0, 0, 2, 0, 0, 0) \]
\[ D_2(G_1, 7) = (0, 0, 1, 0, 0, 2, 0, 0, 0) \]
\[ D_2(G_1, 8) = (2, 0, 0, 0, 0, 1, 0, 0, 0) \]
\[ D_2(G_1, 9) = (0, 0, 1, 0, 0, 0, 0, 0, 0) \]

\[ X_2(G_1) = \{0, 9\}, \{8\}, \{2, 4, 6, 7\}, \{1\}, \{3, 5\}. \]

\[ D_2(G_2, a) = (0, 0, 2, 0, 0, 1, 0, 0, 0) \]
\[ D_2(G_2, b) = (0, 0, 1, 0, 0, 2, 0, 0, 0) \]
\[ D_2(G_2, c) = (0, 0, 1, 0, 0, 2, 0, 0, 0) \]
\[ D_2(G_2, d) = (0, 0, 1, 0, 0, 2, 0, 0, 0) \]
\[ D_2(G_2, e) = (0, 0, 5, 0, 0, 1, 0, 0, 0) \]
\[ D_2(G_2, f) = (0, 0, 5, 0, 0, 1, 0, 0, 0) \]
\[ D_2(G_2, g) = (0, 0, 1, 0, 0, 2, 0, 0, 0) \]
\[ D_2(G_2, h) = (2, 0, 0, 0, 0, 1, 0, 0, 0) \]
\[ D_2(G_2, i) = (0, 0, 1, 0, 0, 0, 0, 0, 0) \]
\[ D_2(G_2, j) = (0, 0, 1, 0, 0, 0, 0, 0, 0) \]

\[ X_2(G_2) = \{i, j\}, \{h\}, \{b, c, d, g\}, \{a\}, \{e, f\}. \]
Computing Graph Isomorphism - Example

This restricts a possible isomorphism to bijections between the following sets:

\[
\begin{align*}
\{0, 9\} & \leftrightarrow \{i, j\} \\
\{8\} & \leftrightarrow \{h\} \\
\{2, 4, 6, 7\} & \leftrightarrow \{b, c, d, g\} \\
\{1\} & \leftrightarrow \{a\} \\
\{3, 5\} & \leftrightarrow \{e, f\}
\end{align*}
\]

There are \(96 = (2!)(1!)(4!)(1!)(2!)\) bijections giving the possible isomorphisms. Examination of each of these possible isomorphisms shows that only the following eight bijections are isomorphisms.
Computing Graph Isomorphism - Example

\[
\begin{align*}
&(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) & (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) \\
&(i, a, d, e, g, f, b, c, h, j) & (j, a, d, e, g, f, b, c, h, i) \\
&(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) & (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) \\
&(i, a, d, e, g, f, c, b, h, j) & (j, a, d, e, g, f, c, b, h, i) \\
&(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) & (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) \\
&(i, a, g, e, d, f, b, c, h, j) & (j, a, g, e, d, f, b, c, h, i) \\
&(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) & (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) \\
&(i, a, g, e, d, f, c, b, h, j) & (j, a, g, e, d, f, c, b, h, i) \\
\end{align*}
\]
Computing Graph Isomorphism

1) **Function** `FINDISOMORPHISM` (set of invariant inducing functions $I$; graph $G_1, G_2$) :

   - set of isomorphisms

2) try {
3)   (partitions, $X, Y$) = GETPARTITIONS ($I, G_1, G_2$);
4) }
5) catch ("$G_1$ and $G_2$ are not isomorphic!") { return $\emptyset$; }
6) for $i = 0$ to partitions - 1 do {
7)   for each $x \in X[i]$ do {
8)     $W[x] = i$;
9)   }
10) }
11) return COLLECTISOMORPHISMS($G_1, G_2, 0, Y, W, f$)
Computing Graph Isomorphism

1) **Function** \textsc{GetPartitions} \((\text{set of invariant inducing functions } I; \text{ graph } G_1; \text{ graph } G_2) : (\text{number of partitions } N, \text{ partitions of } G_1 X, \text{ partitions of } G_2 Y)\)

2) \(N = 1; \ X[0] = \text{vertices of } G_1; \ Y[0] = \text{vertices of } G_2;\)

3) \textbf{for each } \(D \in I \text{ do } \{

4) \(P = N;\)

5) \textbf{for } i = 0 \textbf{ to } P - 1 \textbf{ do } \{

6) \text{Partition } X[i] \text{ into sets } X_1[i], X_2[i], X_3[i], \ldots, X_m[i] \text{ where } x, y \in X_j[i] \iff D(G_1,x) = D(G_1,y);\)

7) \text{Partition } Y[i] \text{ into sets } Y_1[i], Y_2[i], Y_3[i], \ldots, Y_n[i] \text{ where } x, y \in Y_j[i] \iff D(G_2,x) = D(G_2,y);\)

8) \textbf{if } n \neq m \textbf{ then throw exception } \text{“} G_1 \text{ and } G_2 \text{ are not isomorphic!“};\)

9) \text{Order } Y[i] \text{ into sets } Y_1[i], Y_2[i], Y_3[i], \ldots, Y_n[i] \text{ so that }\)

10) \(\forall x \in X[i], \forall y \in Y[i] : D(G_1,x) = D(G_2,y) \iff x \in X_j[i] \text{ and } y \in Y_j[i];\)

11) \textbf{if } ordering is not possible \textbf{ then throw exception } \text{“} G_1 \text{ and } G_2 \text{ are not isomorphic!“};\)

12) \(N = N + m - 1;\)

13) \}\n
14) \text{Reorder the partitions so that: } |X[i]| = |Y[i]| \leq |X[i+1]| = |Y[i+1]| \text{ for } 0 \leq i < N - 1;\)

15) \}\n
16) \textbf{return } (N, X, Y)\)
Computing Graph Isomorphism

Function COLLECTISOMORPHISMS:

1) COLLECTISOMORPHISMS(graph $G_1, G_2$; partition mapping $W$ as starting vertex of $G_1$ $v$; array [vertices of $G_1$] of parititions of $G_2$ $Y$; indices of partitions of $G_1$ $m$; vertices of $G_2$ $n$; current isomorphism $f$ as set of isomorphisms $R$):

2) if $v = \text{number of vertices of } G_1$ then return $\{ f \}$;
3) $R = \emptyset$;
4) $p = W[v]$;
5) for each $y \in Y[p]$ do {
6) \hspace{1em} $OK = \text{true}$;
7) \hspace{1em} for $u = 0$ to $v - 1$ do {
8) \hspace{2em} if $\{ u, v \} \in \text{edges of } G_1 \text{ xor } \{ f[u], y \} \in \text{edges of } G_2$ then $OK = \text{false}$; break;
9) }
10) if $OK$ then {
11) \hspace{1em} $f[v] = y$;
12) \hspace{1em} $R = R \cup \text{COLLECTISOMORPHISMS}(G_1, G_2, v+1, Y, W, f)$;
13) }
14) }
15) return $R$
A certificate $Cert$ for family $\mathcal{F}$ of graphs is a function such that

$$\forall G_1, G_2 \in \mathcal{F} : Cert(G_1) = Cert(G_2) \iff G_1 \text{ is isomorphic to } G_2$$

Currently, the fastest general graph isomorphism algorithms use methods based on computing of certificates.

Computing of certificates works not only for general graphs but it can be also applied on some classes of graphs like trees.
Computing Tree Certificate

1) Label all the vertices of $G$ with the string 01.

2) While there are more than two vertices of $G$ do:
   For each non-leaf $x$ of $G$:
   a) Let $Y$ be the multi-set of labels of the leaves adjacent to $x$ and the label of $x$, with the initial 0 and trailing 1 deleted from $x$;
   b) Replace the label of $x$ with concatenation of the labels in $Y$ sorted in increasing lexicographic order, with 0 prepended and a 1 appended;
   c) Remove all leaves adjacent to $x$.

3) If there is only one vertex left, report the label of $x$ as certificate.

4) If there are two vertices $x$ and $y$ left, then report the labels of $x$ and $y$, concatenated in increasing lexicographic order, as the certificate.
Computing Tree Certificate - Example

number of vertices: 12

non-leaves vertices:

0 : $Y = \emptyset$
1 : $Y = \{01\}$
2 : $Y = \{01,01\}$
5 : $Y = \{01\}$
7 : $Y = \{01\}$
8 : $Y = \{01\}$
Computing Tree Certificate - Example

number of vertices: 6

non-leaves vertices:

\[ Y = \begin{cases} 001011, \\ 0011, \\ 0011 \end{cases} \]

\[ Y = \begin{cases} 0011, \\ 0011 \end{cases} \]
Computing Tree Certificate - Example

number of vertices: 2

Certificate = 0001011001100111000110111

0 : 001011001100111

5 : 0011011
Computing Tree Certificate

- properties of certificate:
  - the length is $2 \cdot |V|$  
  - the number of 1s and 0s is the same  
  - furthermore, the number of 1s and 0s is the same for every partial subsequence that arise from any label of vertex (during the whole run of the algorithm)
Reconstruction of Tree from Certificate - Example

\[
f(0) = 0
\]

\[
f(x + 1) = \begin{cases} 
  f(x) + 1, & \text{if } Cert(G)[x] = 0 \\
  f(x) - 1, & \text{if } Cert(G)[x] = 1
\end{cases}
\]

\[
Cert(G) = 0001011001100111100011011
\]
Reconstruction of Tree from Certificate - Example

\[ \text{Cert}(G) = 0001011001110011100011011 \]
Reconstruction of Tree from Certificate - Example

\[ Cert(G) = 0001011001100111100011011 \]
Reconstruction of Tree from Certificate - Example

\[ f: \]

\[ \text{Cert}(G) = 0001011001100111100011011 \]
Reconstruction of Tree from Certificate

1) Function \textsc{FindSubMountains} (integer \(l\), certificate as string \(C\)) : number of submountines in \(C\)

2) \(k = 0; \ M[0] = \) the empty string; \(f = 0;\)

3) for \(x = l - 1\) to \(|C| - l\) do {
   4) if \(C[x] = 0\) then \{ \(f = f + 1;\) \} else \{ \(f = f - 1;\) \}
   5) \(M[k] = M[k] \cdot C[x];\)
   6) if \(f = 0\) then \{ \(k = k + 1; \ M[k] = \) the empty string; \(f = 0;\) \}

7) }

8) return \(k;\)

1) Function \textsc{CertificateToTree} (certificate as string \(C\)) : tree as \(G = (V, E)\)

2) \(n = \frac{|C|}{2}; \ v = 0; \ (V, E) = \) empty graph of order \(n; \ V = \{0, \ldots, n - 1\};\)

3) \(k = \textsc{FindSubMountains}(1, C);\)

4) if \(k = 1\) then \{ \(Label[v] = M[0]; \ v = v + 1;\) \}

5) else \{ \(Label[v] = M[0]; \ v = v + 1; Label[v] = M[1]; \ v = v + 1; \ E = E \cup \{0,1\};\) \}

6) for \(i = 0\) to \(n - 1\) do {
   7) if \(|Label[i]| > 2\) then {
      8) \(k = \textsc{FindSubMountains}(2, Label[i]); \ Label[i] = "01";\)
   9) for \(j = 0\) to \(k - 1\) do \{ \(Label[v] = M[j]; \ E = E \cup \{(i, v)\}; \ v = v + 1;\) \}

10) }

11) return \(G = (V, E);\)

\(O(|C|^2)\)
Reconstruction of Tree from Certificate

1) **Function** `FAST_CERTIFICATE_TO_TREE` (certificate as string $C$) : tree as $G = (V, E)$

2) $(V, E) =$ empty digraph of order $\frac{|C|}{2}$; $V = \{0, \ldots, \frac{|C|}{2}\}$;

3) $n = 0$;

4) $p = n$;

5) **for** $x = 1$ **to** $|C| - 2$ **do** {

6)  **if** $C[x] = 0$ **then** {

7)      $n = n + 1$;

8)      $E = E \cup \{(p, n)\}$;

9)      $p = n$;

10)  **else** {

11)     $p = parent^+(p)$;

12)  }

13)  }

14)  **return** $G = (V, remove.orientation(E))$;

† $parent(x)$ returns the parent of a node $x$. It returns $x$ in the case where $x$ has no parent.