Title will be here

Rodion Efremov

Master thesis
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, May 7, 2016

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Laitos — Institution — Department Department of Computer Science

Tekijä — Författare — Author Rodion Efremov
Työn nimi — Arbetets titel — Title Title will be here
Oppiaine — Läroämne — Subject Computer Science

Työn laji — Arbetets art — Level Master thesis	Aika — Datum — Month and year May 7, 2016	Sivumäärä — Sidoantal — Number of pages 2

Tiivistelmä — Referat — Abstract

Abstract goes here

Avainsanat — Nyckelord — Keywords

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

Dummy section

0.1 Strongly connected components

Since our methods require the input graph to be strongly connected, we review here briefly how to algorithmically validate that the input graph exhibits the requirement. A strongly connected component is any (maximal) subset of vertices C, such that any node $u \in C$ is reachable from any other node of C. A directed graph $G = (V, A)$ is called *strongly connected* if and only if V is a strongly connected component.

We say that u and v are *mutually reachable* whenever they are reachable from each other. Let $u \overset{r}{\sim} v$ denote the aforementioned reachability relation. Now, it is easy to see that

(Reflexivity) $u \overset{r}{\sim} u$, for all $u \in V(G)$.

(Symmetry) if $u \overset{r}{\sim} v$, then $v \overset{r}{\sim} u$.

(Transitivity) If $u \overset{r}{\sim} v$ and $v \overset{r}{\sim} v'$, then $u \overset{r}{\sim} v'$.

The above three properties imply that $\overset{r}{\sim}$ is an equivalence relation, and as such, implies that the graph has a unique partition into strongly connected components. Note that any node is contained within exactly one strongly connected component.

Algorithms for finding all strongly connected components of a graph in linear time are known. We review three of them below.

0.1.1 Kosaraju's algorithm

Definition 0.1. Given a directed graph $G = (V, A)$, the *transpose* of G is the graph $G^T = (V, A^T)$, where $A^T = \{(v, u) \colon (u, v) \in A\}$.

Theorem 0.2. *Any directed graph $G = (V, A)$ has exactly the same strongly connected components as its transpose $G^T = (V, A^T)$.*

Proof. Let $u, v \in V$ be two distinct nodes of the graph. We need to show that u and v are mutually reachable in G if and only if they are mutually reachable in the transpose graph G^T. Suppose u and v are mutually reachable in G. Now, there are two distinct paths: π_{uv} is an $u - v$ path, and π_{vu} is a $v - u$ path. Taking the transpose of the graph (changing the direction of each arc), π_{uv} becomes a $v - u$ path, and π_{vu} becomes a $u - v$ path. Hence, the two nodes u and v are mutually reachable also in the transpose graph.

Suppose now that u and v are not mutually reachable. This implies that there may exist a $u - v$ path π_{uv}, or a $v - u$ path π_{vu}, but not both. Assume that only π_{uv} exists. Now, in the transpose graph, there is a $v - u$ path, yet no $v - u$ path, and so u and v are not mutually reachable in the transpose. The case of π_{vu} is symmetrical.

Since we assumed the nodes u and v to be arbitrary, the result follows. $\square$

Kosaraju's algorithm sacrifices two depth-first search traversals over the input graph: the first one in forward direction and the second one in backward direction (from a node to its parents). As such, it relies on the fact that any directed graph G has exaclty the same strongly connected components than its transpose G^T, and uses that observation for detecting strongly connected components.

Algorithm 1: KosarajuVisit(G, S, L, v)

```
1 if $v \notin S$ then
2     $S \leftarrow S \cup \{v\}$
3     foreach $(v, w) \in G(A)$ do
4         KosarajuVisit$(G, S, L, w)$
5     $L \leftarrow \langle v \rangle \circ L$
```

Algorithm 2: KosarajuAssign(G, μ, u, r)

```
1 if $u$ is not mapped in $\mu$ then
2     $\mu(u) \leftarrow r$
      For all parents of $u$
3     foreach $(v, u) \in G(A)$ do
4         KosarajuAssign$(G, \mu, v, r)$
```

0.1.2 Tarjan's algorithm

Tarjan's algorithm ([?]) for strongly connected components achieves better running times than Kosaraju's algorithm despite the fact that the former was discovered prior to the latter. It does so by doing only one depth-first traversal over the graph, unlike the Kosaraju's algorithm that requires two.

0.1.3 Path-based algorithm

What comes to path-based strongly connected component algorithms, there are several version of the algorithm, with some of them running in superlinear time. The following algorihtm is due to Gabow [1]:

References

[1] Gabow, Harold N.: *Path-based depth-first search for strong and biconnected components.* Inf. Process. Lett., 74(3-4):107–114, 2000. http://dx.doi.org/10.1016/S0020-0190(00)00051-X.

Algorithm 3: KosarajuSCC(G)

```
1  $S \leftarrow \varnothing$
2  $L \leftarrow \langle\rangle$
3  $\mu \leftarrow \varnothing$
4  foreach $v \in V(G)$ do
5      KosarajuVisit($G, S, L, v$)
   Iterate the list $L$ in its natural order
6  foreach $v \in L$ do
7      KosarajuAssign($G, \mu, v, v$)
8  $f = \varnothing$
9  foreach $(v, i) \in \mu$ do
10     if $i$ is not mapped in $f$ then
11         $f(i) \leftarrow \{v\}$
12     else
13         $f(i) \leftarrow f(i) \cup \{v\}$
14 foreach $(i \mapsto C) \in f$ do
15     output $C$
```

Algorithm 4: TarjanStrongConnect(G, u, i, l, j, S)

```
1  $i(u) \leftarrow j$
2  $l(u) \leftarrow j$
3  $j \leftarrow j + 1$
4  Push($S, u$)
5  foreach $(u, v) \in G(A)$ do
6      if $v$ is not mapped in $i$ then
7          TarjanStrongConnect($G, v, i, l, j, S$)
8          $l(u) \leftarrow \min(l(u), l(v))$
9      else if $v \in S$ then
10         $l(u) \leftarrow \min(l(u), i(v))$
11 if $l(u) = i(u)$ then
12     $C = \varnothing$
13     repeat
14         $w =$ Pop($S$)
15         $C \leftarrow C \cup \{w\}$
16     until $w \neq u$;
17     output $C$
```

Algorithm 5: TarjanSCC(G)

```
S ← ∅
i ← ∅
l ← ∅
j ← 0
foreach u ∈ G(V) do
    if u is not mapped in i then
        TarjanStrongConnect(G, u, i, l, j, S)
```

Algorithm 6: GabowVisit(G, u, c, π, S, P, A)

```
π(u) ← c
c ← c + 1
S ← S ∪ {u}
P ← P ∪ {u}
foreach (u, v) ∈ G(A) do
    if v is not mapped in π then
        GabowVisit(G, v, c, π, S, P, A)
    else if v ∉ A then
        while π(Top(P)) > π(v) do
            Pop(P)
if u = Top(P) then
    Pop(P)
    C ← ∅
    repeat
        w ← Pop(S)
        A ← A ∪ {w}
        C ← C ∪ {c}
    until Top(S) ≠ u;
    output C
```

Algorithm 7: GabowSCC(G)

```
1 $S \leftarrow \varnothing$
2 $P \leftarrow \varnothing$
3 $A \leftarrow \varnothing$
4 $\pi \leftarrow \varnothing$
5 $c \leftarrow 0$
6 foreach $u \in V(G)$ do
7     if $u$ is not mapped in $\pi$ then
8         GabowVisit($G, u, c, \pi, S, P, A$)
```