SCC Detection

ADVANCED ALGORITHMS AND PARALLEL PROGRAMMING

POLITECNICO DI MILANO 2017-2018

https://github.com/andreicap/sccaa

Andrei Capastru 898209
Daniel Ballesteros Castilla 898179

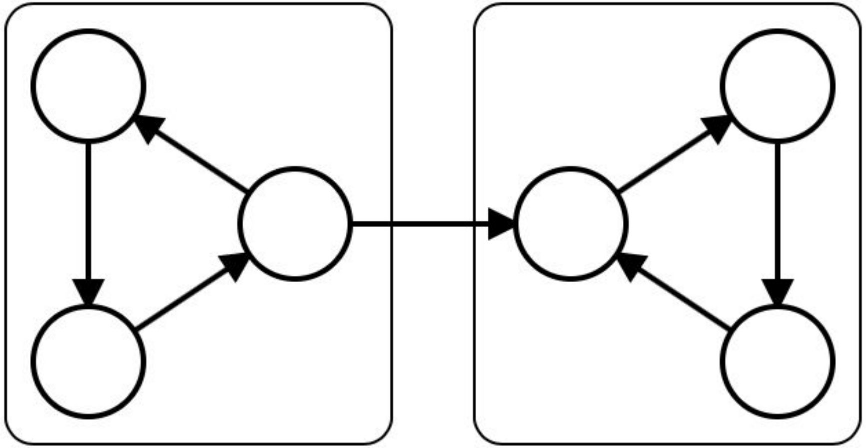
TECHNOLOGIES

C++
Boost Library

ALGORITHMS

Tarjan
Nuutila
Pearce Recursive 1
Pearce Recursive 2

STRONG COMPONENTS



DFS

$index = 0$
for all $v \in V$ **do** $visited[v] = false$
for all $v \in V$ **do**
    if $\neg visited[v]$ **then** visit(v)

procedure visit(v)
$visited[v] = true$; $index = index + 1$
for all $v \to w \in E$ **do**
    if $\neg visited[w]$ **then** visit(w)

o(v+e)
v(1+2w)

TARJAN

(from Nuutilla paper)

v(2+5w)

```
procedure VISIT(v);
begin
    root[v] := v; InComponent[v] := False;
    PUSH(v, stack);
    for each node w such that (v, w) ∈ E do begin
        if w is not already visited then VISIT(w);
        if not InComponent[w] then root[v] := MIN(root[v], root[w])
    end;
    if root[v] = v then
        repeat
            w := POP(stack);
            InComponent[w] := True;
        until w = v
end;
begin/* Main program */
    stack := ∅;
    for each node v ∈ V do
        if v is not already visited then VISIT(v)
end.
```

NUUTILA

```
begin
    root[v] := v; InComponent[v] := False;
    for each node w such that (v, w) ∈ E do begin
        if w is not already visited then VISIT1(w);
        if not InComponent[w] then root[v] := MIN(root[v], root[w])
    end;
    if root[v] = v then begin
        InComponent[v] := True;
        while TOP(stack) > v do begin
            w := POP(stack);
            InComponent[w] := True;
        end
    end else PUSH(v, stack);
end;
begin/* Main program */
    stack := ∅;
    for each node v ∈ V do
        if v is not already visited then VISIT1(v)
end.
```

v(1+4w)

PEARCE 1

v(3+4w)

Variables:
V visited, v visit, vw stack, vw index, v inComponent, v root,

```
for all v ∈ V do visited[v] = false
S = ∅ ; index = 0 ; c = 0
for all v ∈ V do
    if ¬visited[v] then visit(v)
return rindex

procedure visit(v)
root = true ; visited[v] = true                      // root is local variable
rindex[v] = index ; index = index + 1
inComponent[v] = false

for all v → w ∈ E do
    if ¬visited[w] then visit(w)
    if ¬inComponent[w] ∧ rindex[w] < rindex[v] then
        rindex[v] = rindex[w] ; root = false

if root then
    inComponent[v] = true
    while S ≠ ∅ ∧ rindex[v] ≤ rindex[top(S)] do
        w = pop(S)                                   // w in SCC with v
        rindex[w] = c
        inComponent[w] = true
    rindex[v] = c
    c = c + 1
else
    push(S, v)
```

PEARCE 2

v(1+3w)

Variables used:

vw rindex,, index, c v

```
for all v ∈ V do rindex[v] = 0
S = ∅ ; index = 1 ; c = |V| − 1
for all v ∈ V do
        if rindex[v] = 0 then visit(v)
return rindex

procedure visit(v)
root = true                                          // root is local variable
rindex[v] = index ; index = index + 1

for all v → w ∈ E do
        if rindex[w] = 0 then visit(w)
        if rindex[w] < rindex[v] then rindex[v] = rindex[w] ; root = false

if root then
        index = index − 1
        while S ≠ ∅ ∧ rindex[v] ≤ rindex[top(S)] do
                w = pop(S)                           // w in SCC with v
                rindex[w] = c
                index = index − 1
        rindex[v] = c
        c = c − 1
else
        push(S, v)
```

TESTING

Random graph generator BGL
Erdös-Renyi random graph
Own implemented generator

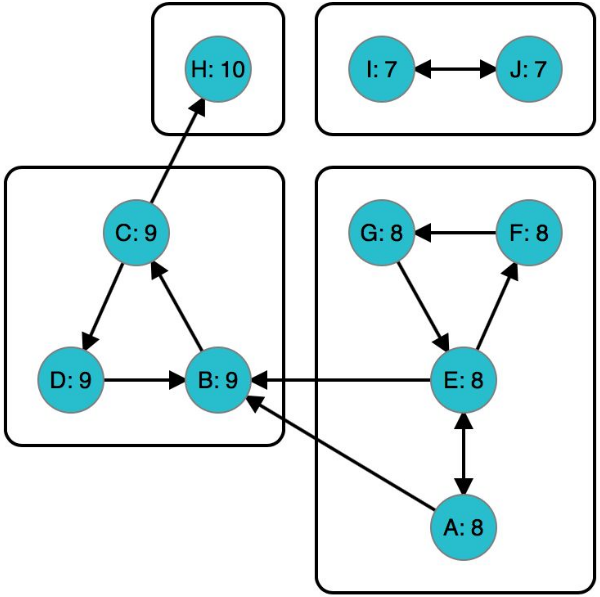
COMPARISON

Tarjan stores all nodes on stack

Nuutila stores only non root nodes

Pearce optimises memory complexity by combining variable

RESULTS




```
Graph ->vertices: 10, edges: 13
0 -> 4
0 -> 1
1 -> 2
2 -> 7
2 -> 3
3 -> 1
4 -> 0
4 -> 1
4 -> 5
5 -> 6
6 -> 4
8 -> 9
9 -> 8
```

Test case: Defined graph
A=0 B=1 C=2 D=3 E=4 F=5 G=6 H=7 I=8 J=9

```
Boost:: Total number of components: 4
Compenent 0: 7
Compenent 1: 1 2 3
Compenent 2: 0 4 5 6
Compenent 3: 8 9

Pearce recursive::components: 4
Component 0: 7
Component 1: 1 2 3
Component 2: 0 4 5 6
Component 3: 8 9

Pearce 2  recursive::components: 4
Component 0: 7
Component 1: 1 2 3
Component 2: 0 4 5 6
Component 3: 8 9

Nuutila ::components: 4
Component: 0 4 5 6
Component: 1 2 3
Component: 7
Component: 8 9

Tarjan recursive::components: 4
Component: 0 4 5 6
Component: 1 2 3
Component: 7
Component: 8 9
```

PERFORMANCE EVALUATION

After evaluating the memory consumption with valgrind(Implemented in MACOS as Instruments) we found important overhead in our first implementation.
We were reallocating memory for all edges at each iteration

4.02 KB	0.0%	2	pearce_recursive_1(int)	pearce1
4.02 KB	0.0%	2	pearce_recursive_1(int)	pearce1
4.02 KB	0.0%	2	pearce_recursive_1(int)	pearce1
4.02 KB	0.0%	2	pearce_recursive_1(int)	pearce1
4.02 KB	0.0%	2	pearce_recursive_1(int)	pearce1
4.02 KB	0.0%	2	pearce_recursive_1(int)	pearce1

Memory allocation for each pearce_recursive call

PERFORMANCE EVALUATION

We used boost reference to access the edges and we got huge improvements:

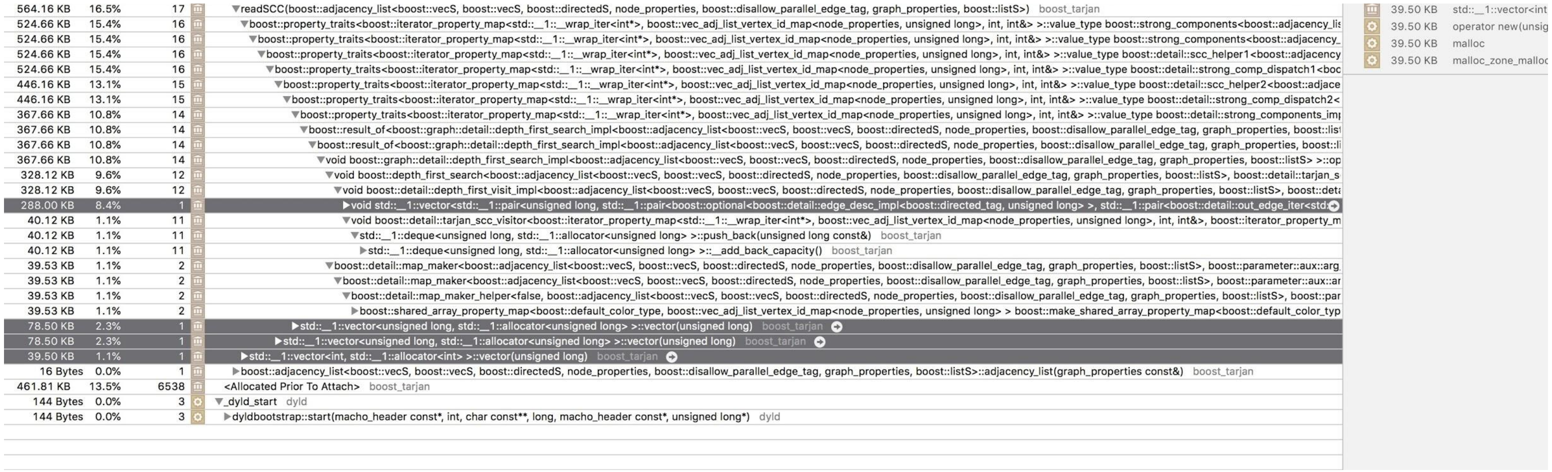
Execution time in pearce1: **from 85s to 6s**
Execution time in pearce2: **from 59s to 1.5s**

16 Bytes	0.0%	1	▼pearce_recursive_2(int)	pearce2
16 Bytes	0.0%	1	▼pearce_recursive_2(int)	pearce2
16 Bytes	0.0%	1	▼pearce_recursive_2(int)	pearce2
16 Bytes	0.0%	1	▼pearce_recursive_2(int)	pearce2
16 Bytes	0.0%	1	▼pearce_recursive_2(int)	pearce2
16 Bytes	0.0%	1	▼pearce_recursive_2(int)	pearce2
16 Bytes	0.0%	1	▼pearce_recursive_2(int)	pearce2
16 Bytes	0.0%	1	▶pearce_recursive_2(int)	pearce2

Memory allocation for each pearce_recursive_2 call

PERFORMANCE EVALUATION

BGL implementation of tarjan algorithms allocates 4 vectors



564.16 KB	16.5%	17	readSCC(boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, node_properties, boost::disallow_parallel_edge_tag, graph_properties, boost::listS>)	boost_tarjan
524.66 KB	15.4%	16	boost::property_traits<boost::iterator_property_map<std::__1::__wrap_iter<int*>, boost::vec_adj_list_vertex_id_map<node_properties, unsigned long>, int, int&> >::value_type boost::strong_components<boost::adjacency_lis	
524.66 KB	15.4%	16	boost::property_traits<boost::iterator_property_map<std::__1::__wrap_iter<int*>, boost::vec_adj_list_vertex_id_map<node_properties, unsigned long>, int, int&> >::value_type boost::strong_components<boost::adjacency_	
524.66 KB	15.4%	16	boost::property_traits<boost::iterator_property_map<std::__1::__wrap_iter<int*>, boost::vec_adj_list_vertex_id_map<node_properties, unsigned long>, int, int&> >::value_type boost::detail::scc_helper1<boost::adjacency	
524.66 KB	15.4%	16	boost::property_traits<boost::iterator_property_map<std::__1::__wrap_iter<int*>, boost::vec_adj_list_vertex_id_map<node_properties, unsigned long>, int, int&> >::value_type boost::detail::strong_comp_dispatch1<boo	
446.16 KB	13.1%	15	boost::property_traits<boost::iterator_property_map<std::__1::__wrap_iter<int*>, boost::vec_adj_list_vertex_id_map<node_properties, unsigned long>, int, int&> >::value_type boost::detail::scc_helper2<boost::adjace	
446.16 KB	13.1%	15	boost::property_traits<boost::iterator_property_map<std::__1::__wrap_iter<int*>, boost::vec_adj_list_vertex_id_map<node_properties, unsigned long>, int, int&> >::value_type boost::detail::strong_comp_dispatch2<	
367.66 KB	10.8%	14	boost::property_traits<boost::iterator_property_map<std::__1::__wrap_iter<int*>, boost::vec_adj_list_vertex_id_map<node_properties, unsigned long>, int, int&> >::value_type boost::detail::strong_components_imp	
367.66 KB	10.8%	14	boost::result_of<boost::graph::detail::depth_first_search_impl<boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, node_properties, boost::disallow_parallel_edge_tag, graph_properties, boost::list	
367.66 KB	10.8%	14	boost::result_of<boost::graph::detail::depth_first_search_impl<boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, node_properties, boost::disallow_parallel_edge_tag, graph_properties, boost::li	
367.66 KB	10.8%	14	void boost::graph::detail::depth_first_search_impl<boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, node_properties, boost::disallow_parallel_edge_tag, graph_properties, boost::listS> >::op	
328.12 KB	9.6%	12	void boost::depth_first_search<boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, node_properties, boost::disallow_parallel_edge_tag, graph_properties, boost::listS>, boost::detail::tarjan_s	
328.12 KB	9.6%	12	void boost::detail::depth_first_visit_impl<boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, node_properties, boost::disallow_parallel_edge_tag, graph_properties, boost::listS>, boost::deta	
288.00 KB	8.4%	1	void std::__1::vector<std::__1::pair<unsigned long, std::__1::pair<boost::optional<boost::detail::edge_desc_impl<boost::directed_tag, unsigned long> >, std::__1::pair<boost::detail::out_edge_iter<std:	
40.12 KB	1.1%	11	void boost::detail::tarjan_scc_visitor<boost::iterator_property_map<std::__1::__wrap_iter<int*>, boost::vec_adj_list_vertex_id_map<node_properties, unsigned long>, int, int&>, boost::iterator_property_m	
40.12 KB	1.1%	11	std::__1::deque<unsigned long, std::__1::allocator<unsigned long> >::push_back(unsigned long const&)	boost_tarjan
40.12 KB	1.1%	11	std::__1::deque<unsigned long, std::__1::allocator<unsigned long> >::__add_back_capacity()	boost_tarjan
39.53 KB	1.1%	2	boost::detail::map_maker<boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, node_properties, boost::disallow_parallel_edge_tag, graph_properties, boost::listS>, boost::parameter::aux::arg	
39.53 KB	1.1%	2	boost::detail::map_maker<boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, node_properties, boost::disallow_parallel_edge_tag, graph_properties, boost::listS>, boost::parameter::aux::ar	
39.53 KB	1.1%	2	boost::detail::map_maker_helper<false, boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, node_properties, boost::disallow_parallel_edge_tag, graph_properties, boost::listS>, boost::par	
39.53 KB	1.1%	2	boost::shared_array_property_map<boost::default_color_type, boost::vec_adj_list_vertex_id_map<node_properties, unsigned long> > boost::make_shared_array_property_map<boost::default_color_typ	
78.50 KB	2.3%	1	std::__1::vector<unsigned long, std::__1::allocator<unsigned long> >::vector(unsigned long)	boost_tarjan
78.50 KB	2.3%	1	std::__1::vector<unsigned long, std::__1::allocator<unsigned long> >::vector(unsigned long)	boost_tarjan
39.50 KB	1.1%	1	std::__1::vector<int, std::__1::allocator<int> >::vector(unsigned long)	boost_tarjan
16 Bytes	0.0%	1	boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, node_properties, boost::disallow_parallel_edge_tag, graph_properties, boost::listS>::adjacency_list(graph_properties const&)	boost_tarjan
461.81 KB	13.5%	6538	<Allocated Prior To Attach>	boost_tarjan
144 Bytes	0.0%	3	_dyld_start	dyld
144 Bytes	0.0%	3	dyldbootstrap::start(macho_header const*, int, char const**, long, macho_header const*, unsigned long*)	dyld

39.50 KB	std::__1::vector<int
39.50 KB	operator new(unsig
39.50 KB	malloc
39.50 KB	malloc_zone_malloc

Figure: Memory allocation for [BGL tarjan algorithm implementation]

PERFORMANCE EVALUATION

2nd Pearce algorithm uses only 1:

39.52 KB	1.6%	2		▼pr2(boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, node_p
39.50 KB	1.6%	1		▼std::__1::vector<int, std::__1::allocator<int> >::resize(unsigned long, int cons
39.50 KB	1.6%	1		▼std::__1::vector<int, std::__1::allocator<int> >::__append(unsigned long, int
39.50 KB	1.6%	1		▼std::__1::__split_buffer<int, std::__1::allocator<int>&>::__split_buffer(unsi
39.50 KB	1.6%	1		▼std::__1::__split_buffer<int, std::__1::allocator<int>&>::__split_buffer(un
39.50 KB	1.6%	1		▶operator new(unsigned long) libc++abi.dylib

Figure: Memory allocation for [BGL tarjan algorithm implementation]

Sources:

Pearce, David. "A Space Efficient Algorithm for Detecting Strongly Connected Components". Information Processing Letters. pp. 47–52, (2015).

Nuutila, Esko. "On Finding the Strongly Connected Components in a Directed Graph". Information Processing Letters. pp. 9–14, (1993).

Tarjan, R. E., "Depth-first search and linear graph algorithms", SIAM Journal on Computing, 1 (2): 146–160, (1972)

C++ reference https://en.cppreference.com/w/cpp/

Boost library documentation https://www.boost.org/doc/libs/1_67_0/

Pearce, David GitHub https://github.com/DavePearce/StronglyConnectedComponents

Pearce SCC algorithms explanation http://www.timl.id.au/SCC

Boost library implementation http://marko-editor.com/articles/graph_connected_components/