Design and Implementation of Efficient Data Types for Static Graphs

Stefan Näher and Oliver Zlotowski
University of Trier, Germany

Structure

- Ideas for the characterisation of graph data types
- Some implementation details of the static graphs
- Using static graphs to speed up algorithms
- Experimental results

Motivation

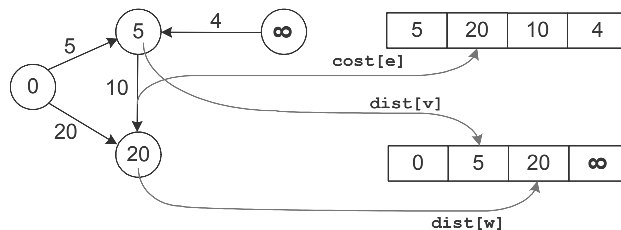
- LEDA contains a graph data type, called **`graph`**, and many other data structures related to graph.
- **`graph`** represents all graphs used in the library, e. g.
  - directed, undirected and bidirected graphs
  - networks
  - planar maps
- **`graph`** provides the operations for all graphs, e. g.

```
G.target(e), G.source(e), G.opposite(e,v),
forall_in_edge(e,u), forall_out_edges(e,u), ...
G.reversal(e), G.make_planar_map(), ...
```

Node and Edge Data

- Node and egde arrays are arrays indexed by the nodes and edges of a graph.

```
node_array<int> dist;
edge_array<int> cost;
dist[w] = dist[v] + cost[e];
```



Example

```
void Dijkstra(const graph& G, node s,
              const edge_array<int>& cost,

{ [I
  [M
}
```

```
[In
node

node
fora

dist
PQ.i
```

```
[Main Loop] ≡
while (!PQ.empty())

      dist[v] = c;
  }
}
```

- Most algorithms don‘t change the graph, i. e., they work on a **static graph**.
- Different algorithms are based on different categories of graphs.

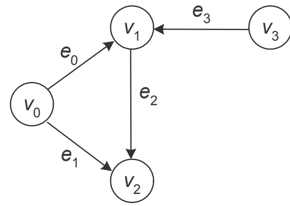
Graph Categories and Structures

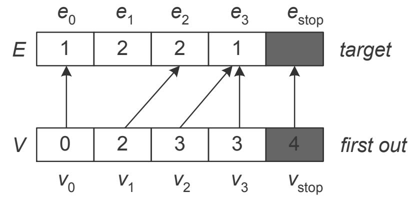
- Structure of a graph describes the sets *V* and *E*
  - doubly- or single-linked lists (dynamic graphs)
  - arrays (static graphs)
- Category of a graph defines the abstract type of the graph
  - directed, bidirectional, and opposite graph
  - bidirected and undirected graph

➢ Graph data type, e. g.

`st_graph<bidirectional_graph>`,
`dl_graph<undirected_graph>`, ...

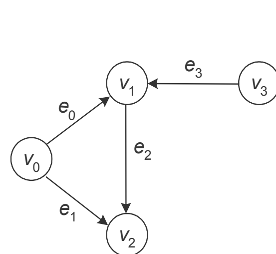
Static Directed Graph

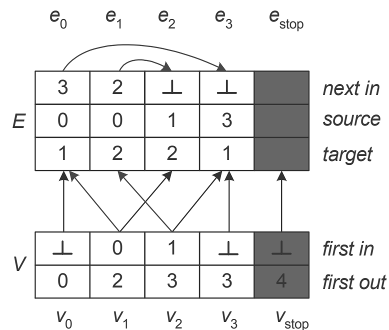





```
forall_nodes(v,G)              for (v = V; v != v_stop; v++)
forall_edges(e,G)              for (e = E; e != e_stop; e++)

st_graph<directed_graph> G

node   G.target(edge e)        e->target
forall_out_edges(e,v)          for (e = v->first_out;
                                    e != (v+1)->first_out; e++)
```

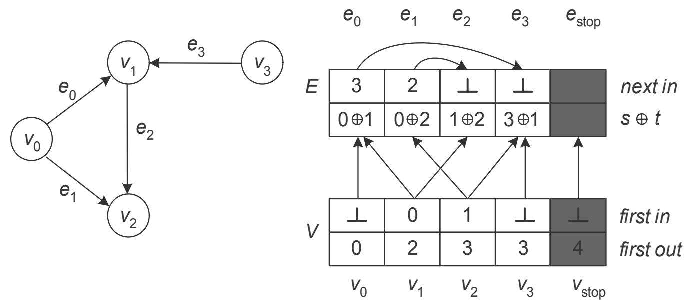
Static Bidirectional Graph






```
st_graph<bidirectional_graph> G
node   G.source(edge e)      e->source
forall_in_edges(e,v)         for (e = v->first_in;
                                  e != 0; e = e->next_in)
```

Static Opposite Graph

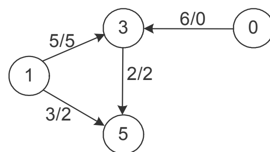



```
st_graph<opposite_graph> G
node   G.opposite(edge e,node v)   e->src_xor_tar ⊕ v
```

Node and Edge Data

- External arrays are realized by node- and edge arrays.
- Static data slot assignment is a method for storing additional information in the edge or node structure.

```
st_graph<bidirectional_graph, data_slots<1>, data_slots<2> >
edge_slot<int,1> flow;
edge_slot<int,0> cap;
node_slot<int,0> excess;
```



5	2	2	0		**flow**
5	3	2	6		**cap**
3	2	⊥	⊥		*next in*
0	0	1	3		*source*
1	2	2	1		*target*

1	5	3	0		**excess**
⊥	0	1	⊥	⊥	*first in*
0	2	3	3	4	*first out*

Graph Algorithms

Dijkstra function template

```
template <class graph_type, class cost_array, class dist_array>
void Dijkstra(const graph_type& G, graph_type::node s,
              const cost_array& cost, dist_array& dist)
{ [Initialization]
  [Main Loop]
}
```

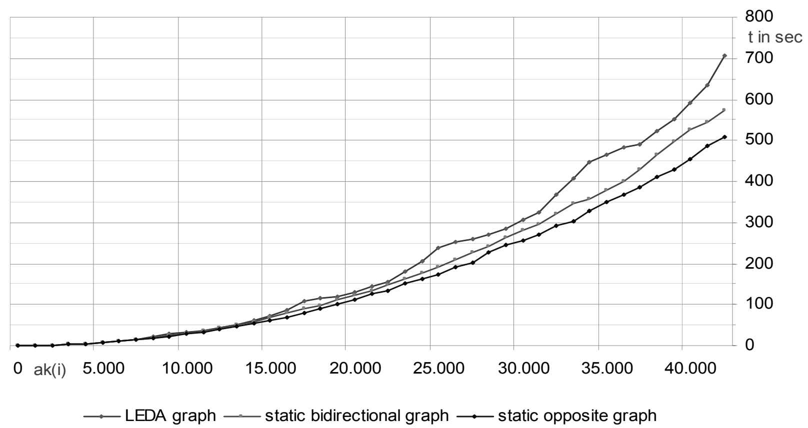
Using the function template in the application

```
typedef st_graph<directed_graph,data_slot<1>,data_slot<1> sgraph;

sgraph G;
sgraph::node s;
edge_slot<int,0,sgraph> cost;
node_slot<int,0,sgraph> dist;

read_dimacs_sp(cin,G,s,cost);
Dijkstra(G,s,cost,dist);
```

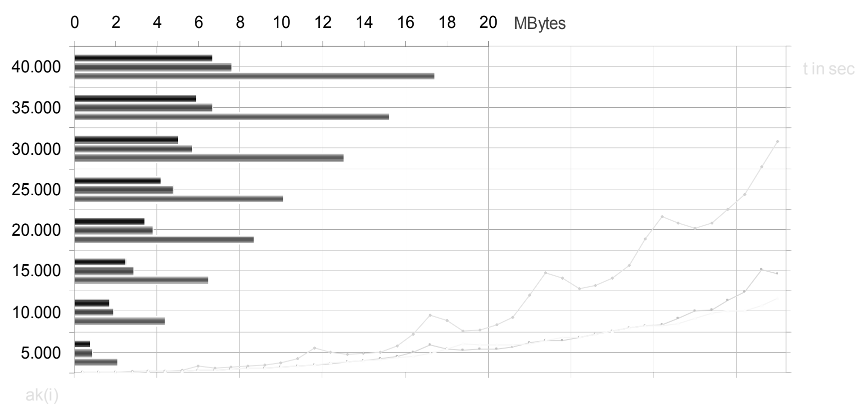
Experiments

- Algorithm
  - Preflow-push maxflow algorithm
  - using gap-heuristic and highest level queue
- Networks
  - *ak*-generator by Cherkassky and Goldberg
  - *ak*(*i*)-network consists of 4*i* + 6 nodes, 6*i* + 7 edges
  - hard maxflow instances for preflow-push algorithm
- Platform
  - Sun UltraSPARC IIi 440 Mhz, 256 Mbytes, Solaris 2.7
  - programs have been compiled with gcc-3.0.3

External Arrays



Static Slot Assignment



Summary and Conclusion

- We presented a new collection of static graph types, which fit into the LEDA environment of graph algorithms.
- The experiments show, that the use of static graphs in combination with efficient methods for the data access speeds up algorithms considerably.
- All users of LEDA can benefit from the presented improvements.