Exploring Generic Heterogeneous Implementations of Graph Algorithms

Projecto Integrado CPD

Cristiano Sousa
Department of Informatics
University of Minho

Artur Mariano
Institute for Scientific Computing
Technische Universität Darmstadt

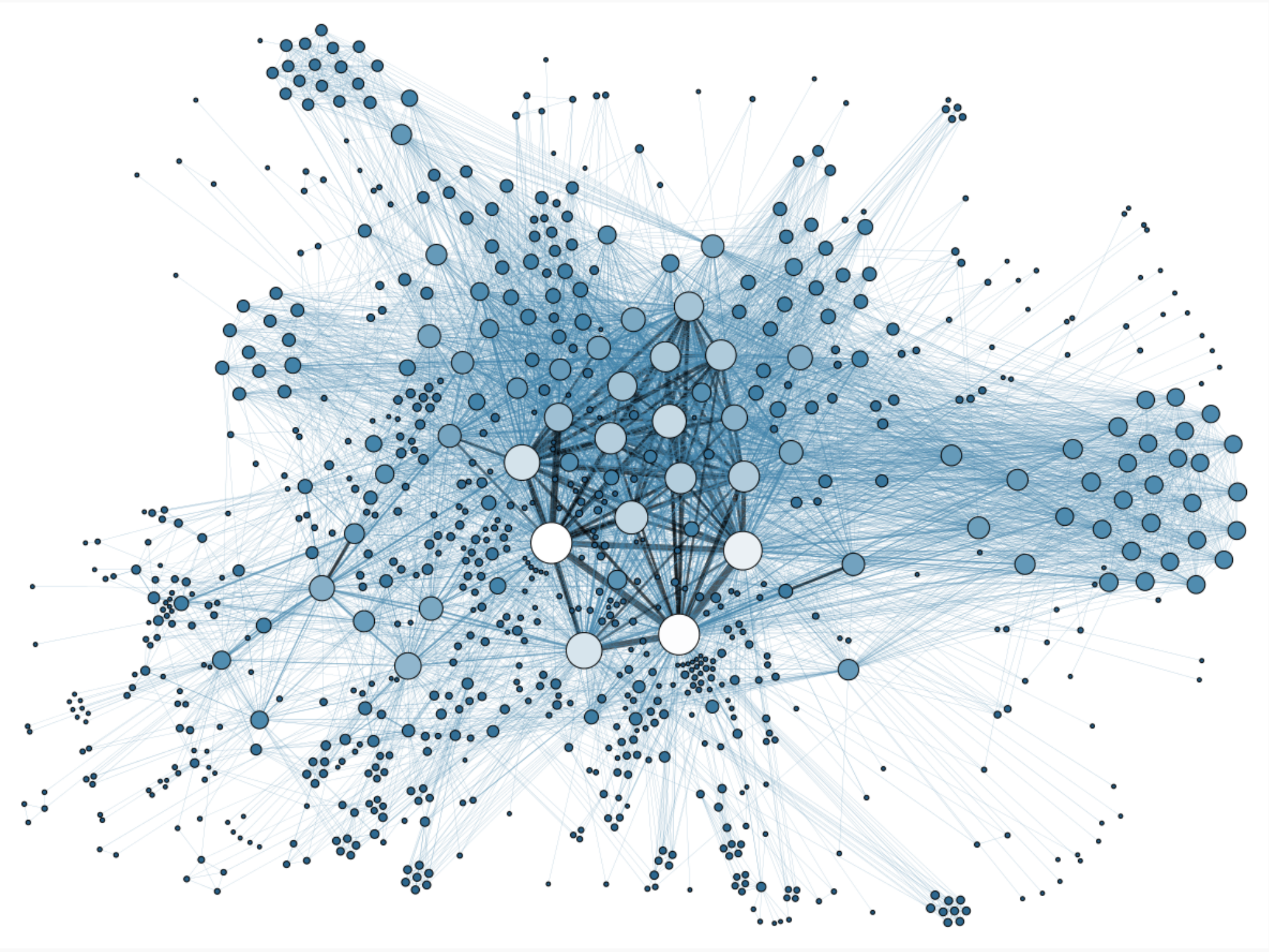
Universidade do Minho

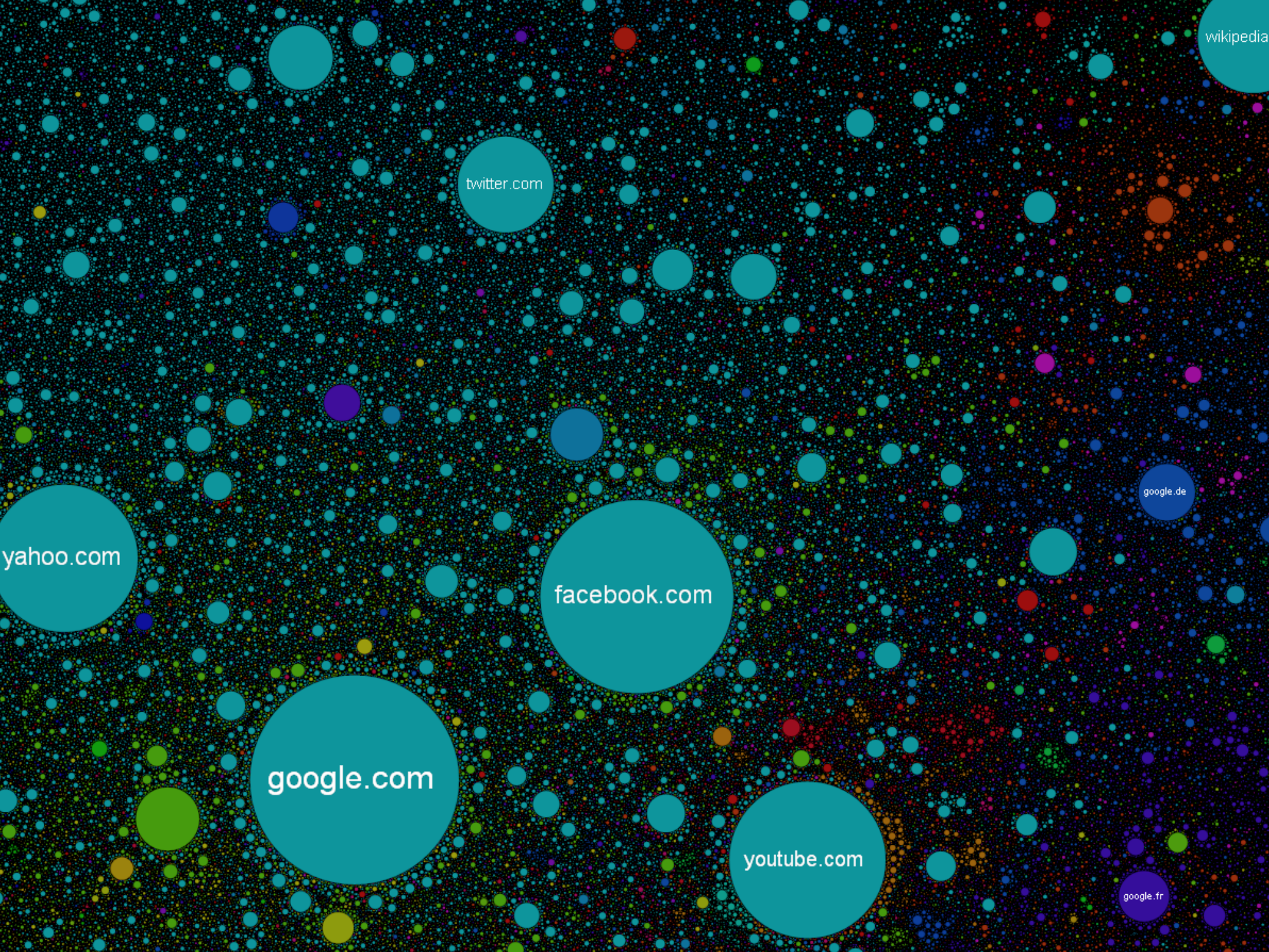
December 19, 2014

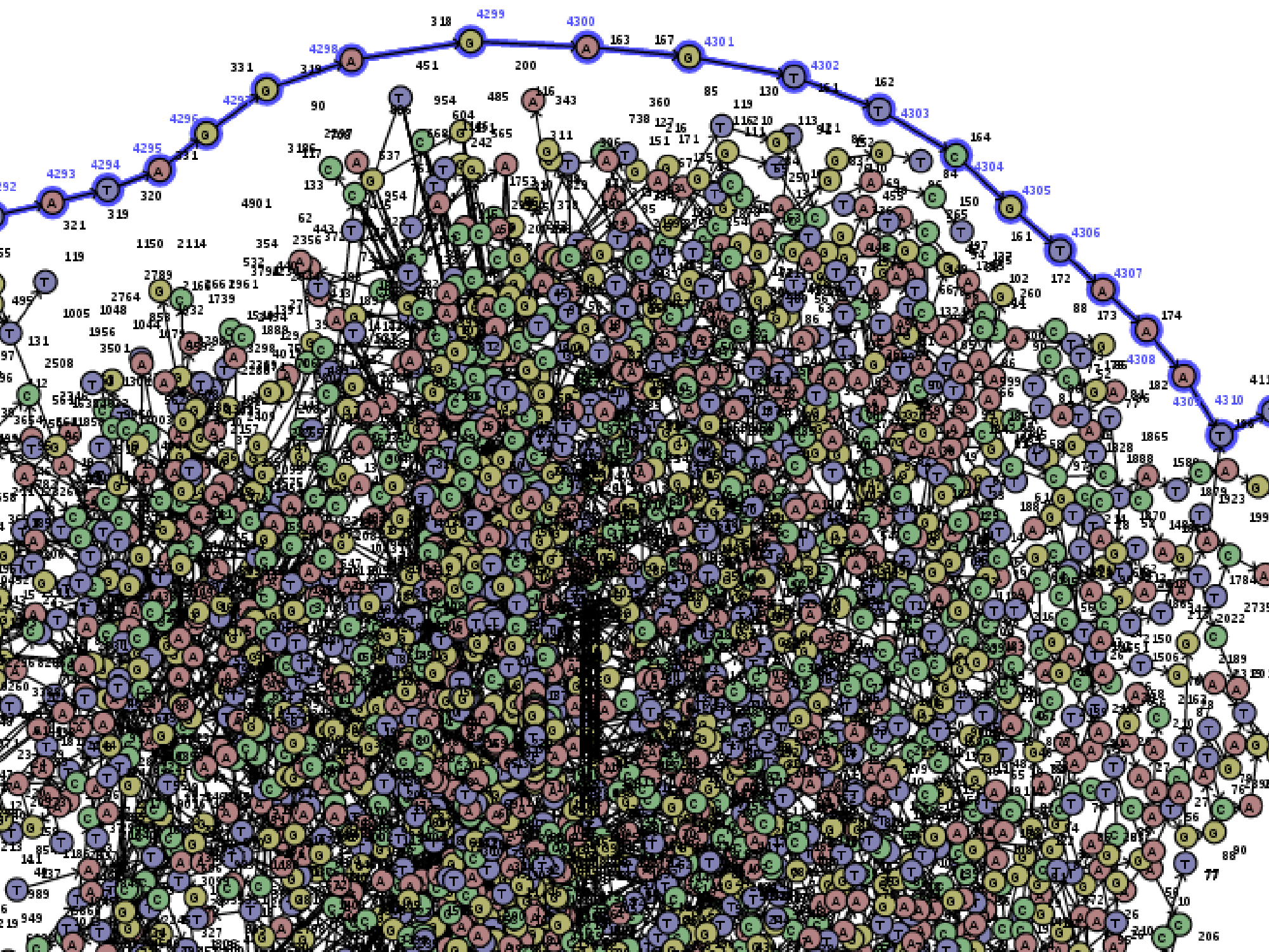
What are Graphs?

- Set of vertices that can be connected through edges
- Vertices can represent any object or entity
- Edges define any type of relationship between vertices

Graphs are important data structures, used in real-world problems




wikipedia
twitter.com
google.de
yahoo.com
facebook.com
google.com
youtube.com
google.fr


4299
4300
4301
4302
4303
4304
4305
4306
4307
4308
4309
4310
4298
4297
4296
4295
4294
4293

Graph Algorithms In Current Problems

- Graph Partitioning
  - To deal with work-load distribution of large graphs
- Minimum Spanning Tree
  - DNA sequencing, Electroencephalographies and other medical studies
  - Used in the approximation of other graph algorithms
- Betweenness Centrality
  - Metrics on any network represented as a graph, to provide QoS, etc.
  - Social network analysis

The challenge:

- (Memory)-Irregular class of algorithms
  - Why?

Adjacency Matrix

- Value at position (i, j) represents an edge from i to j with the specified weight

	a	b	c	d	e	f	g
a		4		12	9		
b	4		8		7		
c		8			5		2
d	12				3	6	
e	9	7	5	3		1	13
f				6	1		11
g			2		13	11	

Figure: Adjacency matrix representation.

- Pros: $\mathcal{O}(1)$ read and insertion and removal of edges
- Cons: $\mathcal{O}(|V| \times |V|)$ memory requirements, $\mathcal{O}(|V|)$ edge iteration per vertex, stores non-existing edges

Adjacency Matrix

- Value at position (i, j) represents an edge from i to j with the specified weight

	a	b	c	d	e	f	g
a		4		12	9		
b	4		8		7		
c		8			5		2
d	12				3	6	
e	9	7	5	3		1	13
f				6	1		11
g			2		13	11	

Figure: Adjacency matrix representation.

Memory usage for large graphs

For example, a graph with 100.000 vertices would use $100.000^2 \times 4$ bytes $\approx$ 37GB of RAM

Adjacency List

- Array of vertices
- Each entry points to a list of destination vertex and weight pairs

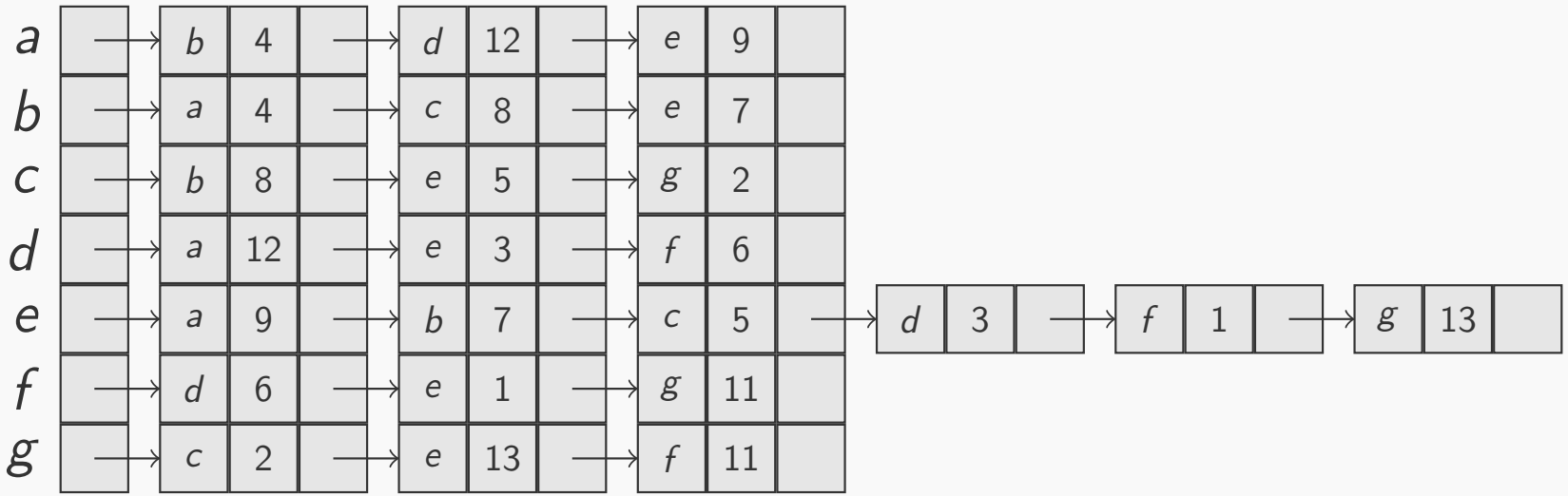


Figure: Adjacency list representation.

- Pros: $\mathcal{O}(1)$ insertion of edges, mutable, efficient edge iteration
- Cons: Heavy usage of linked-lists and pointers, costly edge lookup

Adjacency List

- Array of vertices
- Each entry points to a list of destination vertex and weight pairs

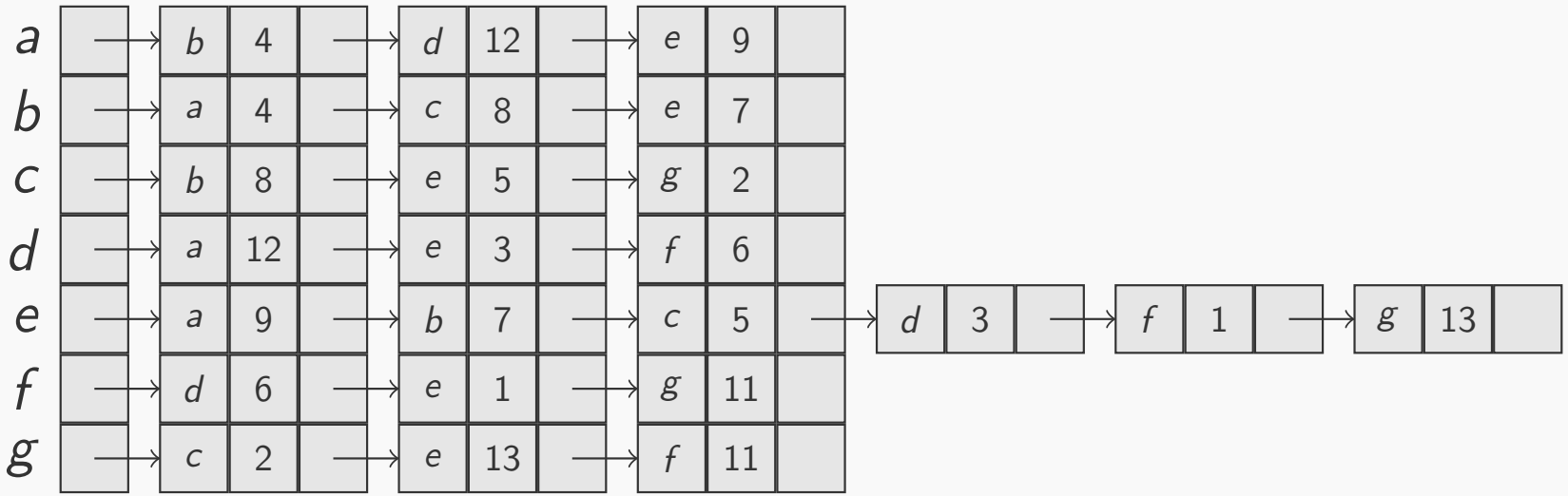


Figure: Adjacency list representation.

Linked-lists?

Instead of linked-lists, arrays can be used, sacrificing mutability for a more cache friendly approach.

Generic Approach

How can we obtain a generic solution?

Generic Approach

How can we obtain a generic solution?

- Graph representation
  - Adjacency Lists are not suited for the GPU
  - Large memory requirements of the Adjacency Matrix limit graph sizes

Generic Approach

How can we obtain a generic solution?

- Graph representation
  - Adjacency Lists are not suited for the GPU
  - Large memory requirements of the Adjacency Matrix limit graph sizes

$\hookrightarrow$ Unified graph representation

Generic Approach

How can we obtain a generic solution?

- Algorithm
  - Different platforms often imply distinct implementations and parallelization approaches
  - This might discourage an heterogeneous implementation

Generic Approach

How can we obtain a generic solution?

- Algorithm
  - Different platforms often imply distinct implementations and parallelization approaches
  - This might discourage an heterogeneous implementation

$\hookrightarrow$ Generic algorithm

and respective platform-independent implementation

Generic Approach

How can we obtain a generic solution?

- Scheduling and Parallelization
  - Usual route of implementing an algorithm and then parallelizing computationally intensive routines is a no go
  - Algorithms must be conceptualized for heterogeneous implementations, from the ground up

Generic Approach

How can we obtain a generic solution?

- Scheduling and Parallelization
  - Usual route of implementing an algorithm and then parallelizing computationally intensive routines is a no go
  - Algorithms must be conceptualized for heterogeneous implementations, from the ground up

$\hookrightarrow$ Cross-platform parallelization and scheduling approach

Generic Approach

How can we obtain a generic solution?

↪ Unified graph representation

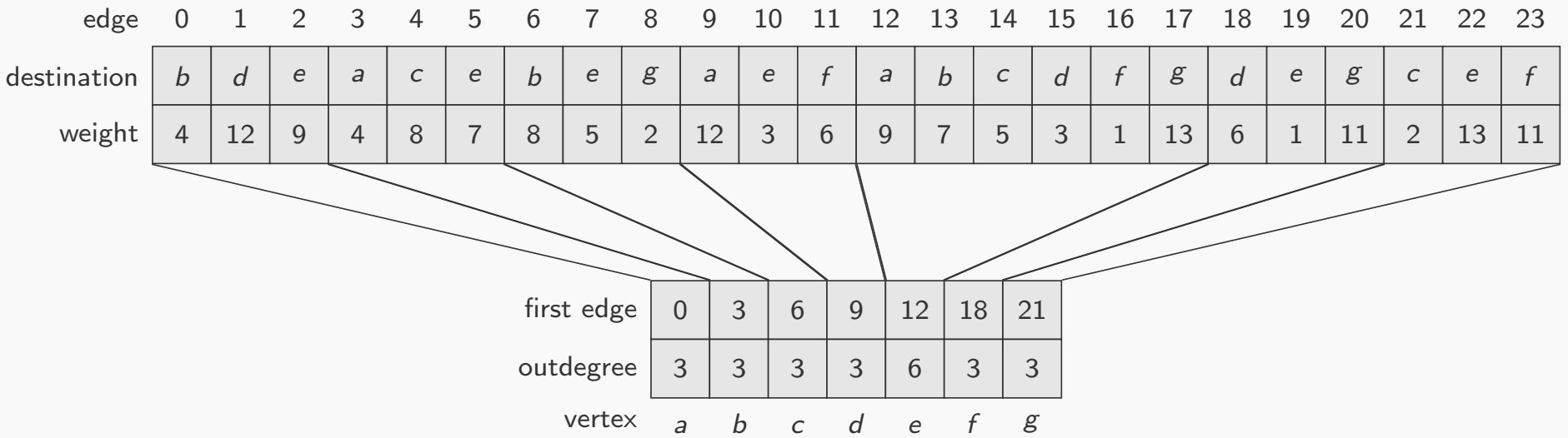
↪ Generic algorithm
and respective platform-independent implementation

↪ Cross-platform parallelization and scheduling approach

Unified Graph Representation

Compressed Sparse Row (CSR) format

- Structure of Arrays (SoA)
- Used to represent sparse matrices
- Often seen in the literature as the representation of choice for graph algorithms on the GPU



Generic Algorithm and Scheduling

Traditional approach:

```
1: while there is work to be done do
2: |   for each work item do
3: |   |   Do stuff...
4: |   Some other stuff
5: |   ...
6: |   for each work item do
7: |   |   Do more stuff...
```

Generic Algorithm and Scheduling

- Topological scheduling: each vertex is a work unit
- Operator based parallelization: operators are applied to each vertex
  - The operator has to be a well defined set of instructions that can be applied to all active vertices, at the same time

Generic Algorithm and Scheduling

- Topological scheduling: each vertex is a work unit
- Operator based parallelization: operators are applied to each vertex
  - The operator has to be a well defined set of instructions that can be applied to all active vertices, at the same time

---

1: **while** there are active vertices **do**
2: | Apply **operator** 1
3: | Synchronize
4: | Apply **operator** 2
5: | Synchronize
6: | ...
7: | Apply **operator** N
8: | Synchronize
9: | Determine active vertices

Generic Algorithm and Scheduling

- Topological scheduling: each vertex is a work unit
- Operator based parallelization: operators are applied to each vertex
  - The operator has to be a well defined set of instructions that can be applied to all active vertices, at the same time

---

```
1: while there are active vertices do
2: |   Apply operator 1
3: |   Synchronize
4: |   Apply operator 2
5: |   Synchronize
6: |   ...
7: |   Apply operator N
8: |   Synchronize
9: |   Determine active vertices
```

Generic Implementation

The body of the operator should be the same across multiple computing platforms. Only synchronization should differ.

The Minimum Spanning Tree Problem

- Minimum Spanning Tree?
  - Well known graph problem
  - Used in Medicine, Biomedics, social-, road- and compute-networks
- To solve the MST-problem, several algorithms have been proposed
  - Borůvka, Kruskal and Prim
- Sequential implementations focus on data structures
- Current parallel implementations are too demanding and have limitations
- (Memory)-Irregular class of algorithms

Example Minimum Spanning Tree

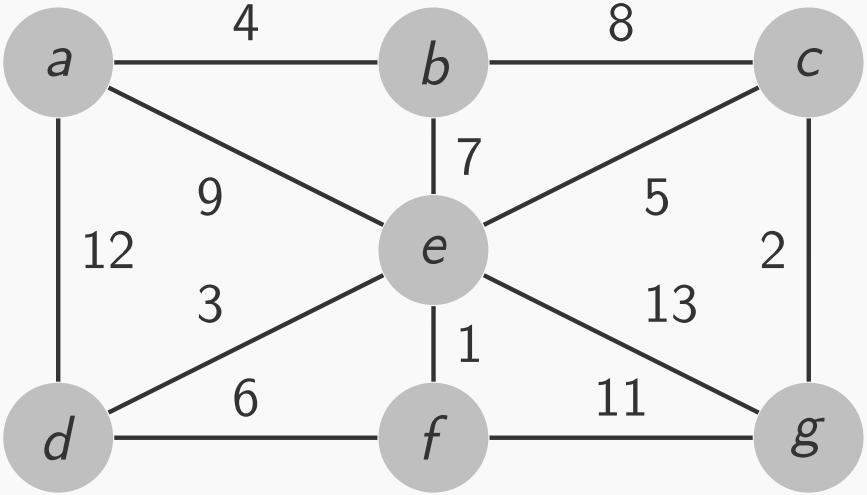


Figure: Example graph.

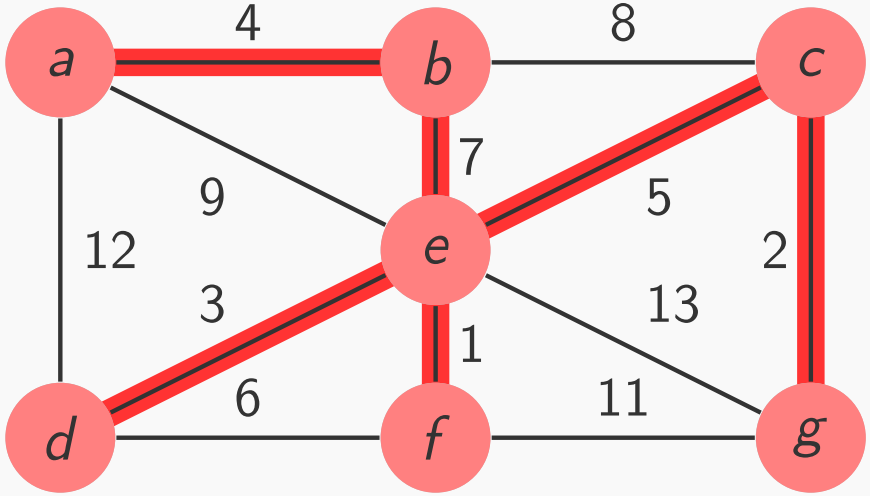


Figure: MST of the example graph.

Borůvka's Algorithm

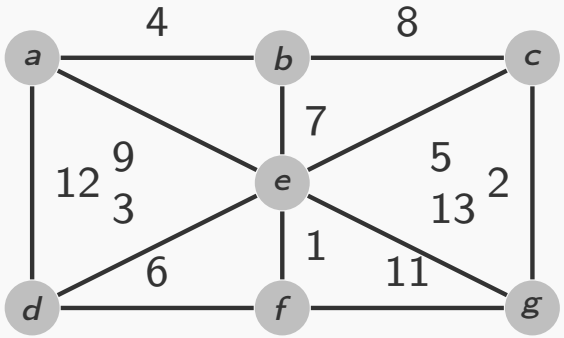


Figure: Initial graph.

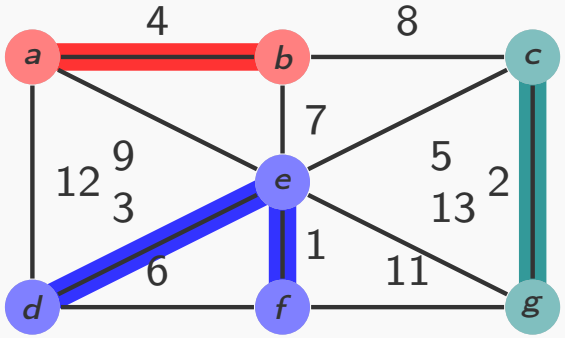


Figure: Find lightest edges.

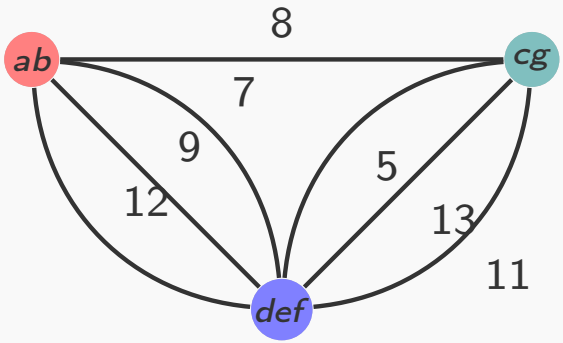


Figure: Contract.

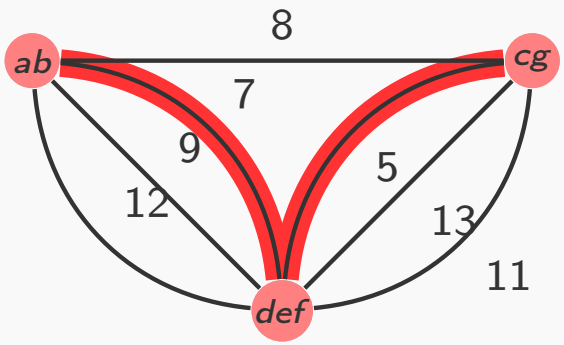


Figure: Find lightest edges.

Figure: Contract.

Algorithm 1 Parallel Generic Borůvka

Input: Undirected, connected and weighted graph $G(V, E)$

```
1:  while number of vertices > 1 do
2:  |   Find minimum edge per vertex
3:  |   Remove mirrored edges
4:  |   Initialize colors
5:  |   while not converged do
6:  |   |   Propagate colors
7:  |   Create new vertex ids
8:  |   Count new edges
9:  |   Assign edge segments to new vertices
10: |   Insert new edges
```

Project Proposal

What will you be working on?

- Assess the performance of the Generic Borůvka implementation
- Explore the generic approach to other graph algorithms
- Library for high performance, generic, heterogeneous, graph algorithms

Resources:

- MSc. Thesis
- Publication
- Generic Borůvka source code
- Artur and me

Exploring Generic Heterogeneous Implementations of Graph Algorithms

Projecto Integrado CPD

Cristiano Sousa
Department of Informatics
University of Minho

Artur Mariano
Institute for Scientific Computing
Technische Universität Darmstadt

Universidade do Minho

December 19, 2014