One-sided Crossing Minimization Problem: Directed Feedback Arc Set Ansatz

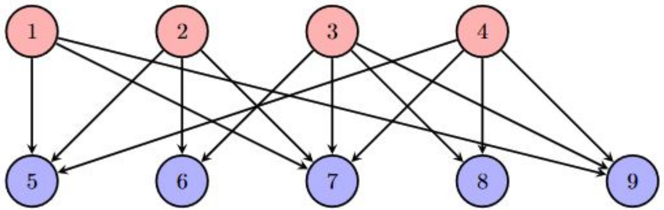
Jonas Tolkemitt
Chanh Quach

Inhalt

1. Das OCM Problem und die Reduktion zu DFAS
2. Beispiel für die Reduktion
3. Übersicht zum Code
4. Evaluierung
5. Stärken und Schwächen

Das OCM Problem und die Reduktion zu DFAS

- One-sided Crossing Minimization Problem (OCM)
  - Ein bipartiter Graph
  - 2 parallel horizontale Reihenfolgen von Knoten
  - Minimale Anzahl von Crossings

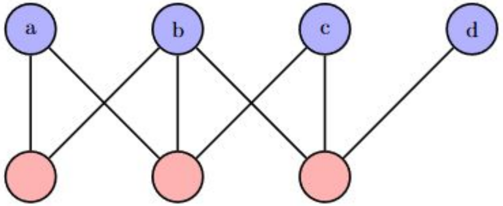


Das OCM Problem und die Reduktion zu DFAS

- Directed Feedback Arc Set (DFAS)
  - Eine Teilmenge von Kanten in einem gerichteten Graphen zu identifizieren, die entfernt werden kann, um den Graphen azyklisch zu machen
  - Das Ziel ist es, eine möglichst kleine Menge von Kanten zu entfernen
- Reduktion
  - Bipartiter Graph zu Penalty Digraph
  - Methode: Penalty Minimization Approach

Beispiel für die Reduktion

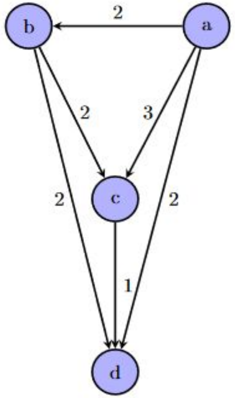
❖ Penalty Minimization Approach



Der original OCM Graph

	a	b	c	d
a	-	1	0	0
b	3	-	1	0
c	3	3	-	0
d	2	2	1	-

Die Tabelle für die Umsortierung und die Anzahl von Crossings



Der Penalty Digraph

Übersicht zum Code

```rust
#[derive(Debug)]
3 implementations
pub struct Graph {
    number_of_nodes: usize,
    number_of_fixed_nodes: usize,
    number_of_free_nodes: usize,
    number_of_edges: usize,
    adjacency_list: Vec<BTreeSet<usize>>,
}
```

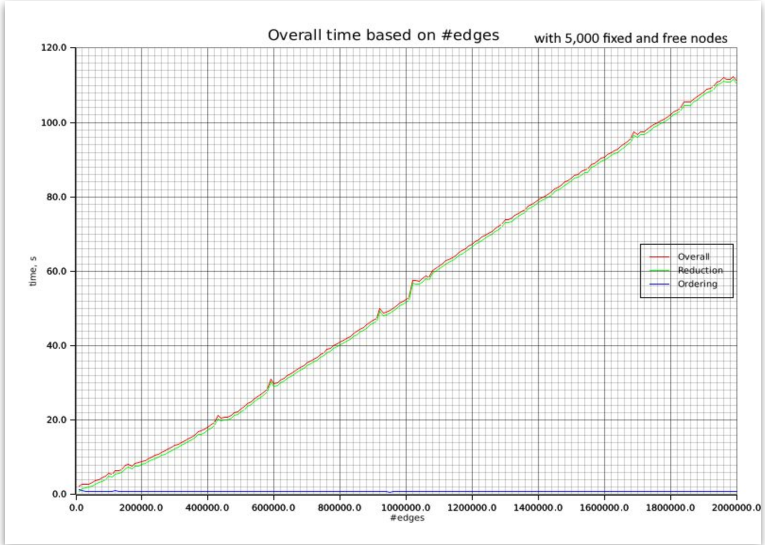
```rust
#[derive(Debug)]
3 implementations
pub struct PenaltyDigraph {
    number_of_nodes: usize,
    adjacency_list: Vec<HashSet<usize>>,
}
```

Übersicht zum Code

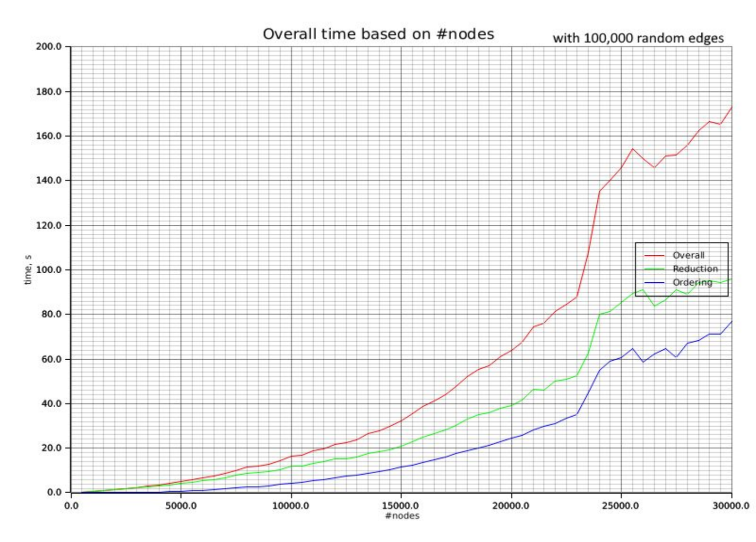
```rust
pub fn sort_fas(&self) -> Vec<usize> {
    let mut feedback_arc_set: Vec<usize> = Vec::new();
    for u: usize in 0..self.number_of_nodes {
        let mut val: isize = 0;
        let mut min: isize = 0;
        let mut loc: usize = u;

        for j: usize in (0..loc).rev() {
            let v: &usize = feedback_arc_set.get(index: j).expect(msg: "Index exists");
            if self.edge_exists(u, *v) {
                val += 1;
            }
            if self.edge_exists(u: *v, v: u) {
                val -= 1;
            }

            if val <= min {
                min = val;
                loc = j;
            }
        }
        feedback_arc_set.insert(index: loc, element: u);
    }

    feedback_arc_set
} fn sort_fas
```

```rust
pub fn from_graph(graph: &Graph) -> PenaltyDigraph {
    let mut penalty_digraph: PenaltyDigraph = PenaltyDigraph::new(number_of_nodes: graph.number_of_free_nodes);

    for u: usize in graph.number_of_fixed_nodes..graph.number_of_nodes {
        for v: usize in u + 1..graph.number_of_nodes {
            let mut c_uv: isize = 0;
            let mut c_vu: isize = 0;
            let mut scan: isize;
            let degree_v: isize = graph.adjacency_list.get(index: v).expect(msg: "Must exist").len() as isize;

            let mut adj_u_iter: Iter<'_, usize> = graph.adjacency_list.get(index: u).unwrap().iter();
            let mut adj_u: Option<&usize> = adj_u_iter.next();

            let mut adj_v_iter: Iter<'_, usize> = graph.adjacency_list.get(index: v).unwrap().iter();
            let mut adj_v: Option<&usize> = adj_v_iter.next();

            while adj_u.is_some() {
                scan = 0;
                while adj_v.is_some() && adj_v.unwrap() < adj_u.unwrap() {
                    adj_v = adj_v_iter.next();
                    scan += 1;
                }
                c_uv += scan;
                c_vu = c_vu + degree_v - scan - 1;

                if adj_u < adj_v {
                    c_vu += 1;
                }
                adj_u = adj_u_iter.next();
            }

            penalty_digraph.add_crossings(
                u: u - graph.number_of_fixed_nodes,
                v: v - graph.number_of_fixed_nodes,
                c_uv,
                c_vu,
            );
        }
    }

    penalty_digraph
} fn from_graph
```

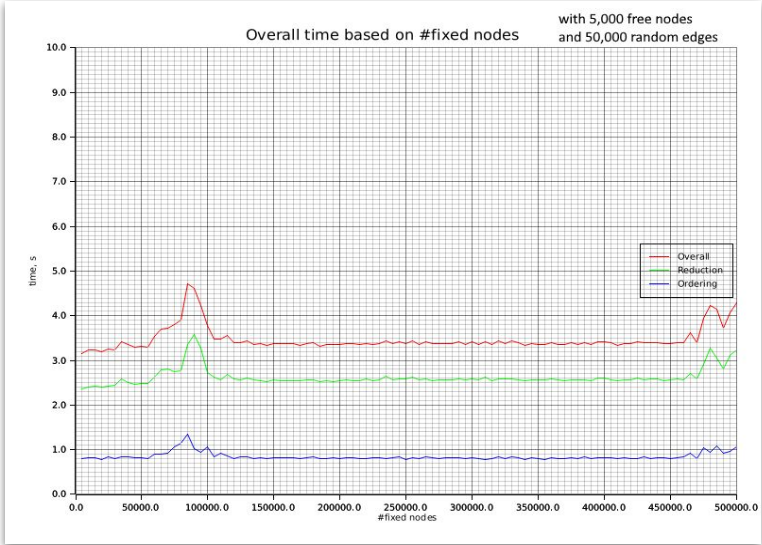
Evaluierung



Evaluierung



Evaluierung



Stärken und Schwächen

- Schlecht für Graphen mit hoher Anzahl an freien Knoten
  - Gut für Graphen, die wenig freie Knoten, aber viele feste Knoten haben
- Insertionsort kann für lokale Optima sorgen
- Reduktion abhängig von Kantenanzahl