Presentation 2

Fynn Lohren, Carsten Schubert, Leon Suchy

November 14, 2018

Structure

- Organisation
- Data Structures
- Lower Bounds
- Data Reduction

Organisation

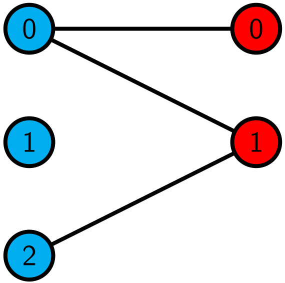
- Issues on Gitlab to track open tasks
- Meeting with milestone goals
- Sick member
  $\rightarrow$ Less time than we wanted to have

Data Structures

- Heaps
- Bipartite Graph
- Controller

Bipartite Graph

Index	Vertex ID	ID left	ID right
0	0	0	
1	uint_max - 1		0
2	1	1	
3	uint_max - 2		1
4	2	2	
5	NULL		-



Controller

- Too much complexity synchronising all data structures
- Model/View/Controller pattern

Clique Cover LB Algorithm

```
procedure CLIQUE COVER(G(V, E))
    clique_set ← {}
    V ← sort_by_degree(V)                          ▷ Sort ascending
    for all v ∈ V do
        clique ← largest_clique(v)       ▷ Largest clique v fits in
        clique_set ← clique_set \ clique
        clique ← clique ∪ {v}
        clique_set ← clique_set ∪ clique
    return clique_set
```

Sorting can be computed in $\mathcal{O}(n \log n)$ time

Clique Cover LB Algorithm

procedure CLIQUE COVER($G(V, E)$)
    $clique\_set \leftarrow \{\}$
    $V \leftarrow sort\_by\_degree(V)$ ▷ Sort ascending
    for all $v \in V$ **do**
        $clique \leftarrow largest\_clique(v)$ ▷ Largest clique v fits in
        $clique\_set \leftarrow clique\_set \setminus clique$
        $clique \leftarrow clique \cup \{v\}$
        $clique\_set \leftarrow clique\_set \cup clique$
    return $clique\_set$

Sorting can be computed in $\mathcal{O}(n \log n)$ time
Finding the largest clique for each v can be done in $\mathcal{O}(deg(v))$ time

Clique Cover LB Algorithm

```
procedure CLIQUE COVER(G(V, E))
    clique_set ← {}
    V ← sort_by_degree(V)                      ▷ Sort ascending
    for all v ∈ V do
        clique ← largest_clique(v)             ▷ Largest clique v fits in
        clique_set ← clique_set \ clique
        clique ← clique ∪ {v}
        clique_set ← clique_set ∪ clique
    return clique_set
```

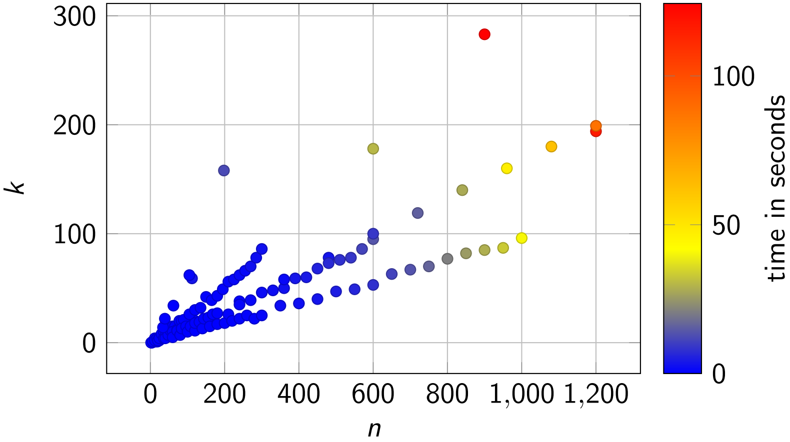
Sorting can be computed in $\mathcal{O}(n \log n)$ time
Finding the largest clique for each v can be done in $\mathcal{O}(deg(v))$ time
More realistic in $\mathcal{O}(\log n)$ time

Clique Cover LB Algorithm

procedure $\textsc{Clique Cover}(G(V, E))$
- $clique\_set \leftarrow \{\}$
- $V \leftarrow sort\_by\_degree(V)$ ▷ Sort ascending
- **for all** $v \in V$ **do**
  - $clique \leftarrow largest\_clique(v)$ ▷ Largest clique v fits in
  - $clique\_set \leftarrow clique\_set \setminus clique$
  - $clique \leftarrow clique \cup \{v\}$
  - $clique\_set \leftarrow clique\_set \cup clique$
- **return** $clique\_set$

Sorting can be computed in $\mathcal{O}(n \log n)$ time
Finding the largest clique for each v can be done in $\mathcal{O}(deg(v))$ time
More realistic in $\mathcal{O}(\log n)$ time
Total running time:
$\mathcal{O}(n \log n)$

Clique Cover LB



Linear Programming LB Algorithm

Hopcroft-Karp: Finds maximum matching in a bipartite graph

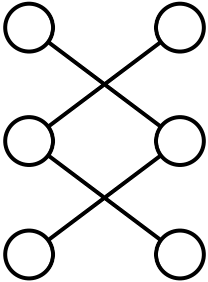
Linear Programming LB Algorithm

Hopcroft-Karp: Finds maximum matching in a bipartite graph
Achieved complexity: $\mathcal{O}(m * \sqrt{n})$

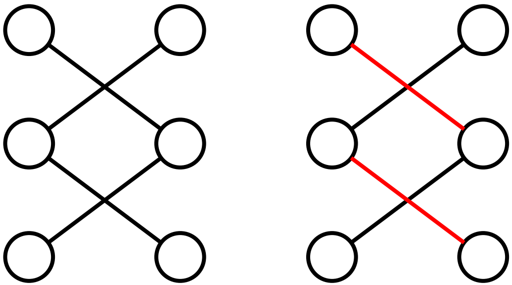
Linear Programming LB Algorithm

Hopcroft-Karp: Finds maximum matching in a bipartite graph
Achieved complexity: $\mathcal{O}(m * \sqrt{n})$
König theorem:
Constructs minimum vertex cover from maximum matching in bipartite graph
Theoretical complexity: $\mathcal{O}(m * \sqrt{n})$

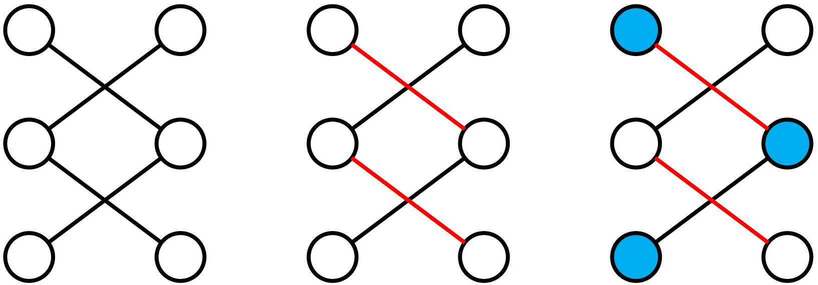
LP bound construction



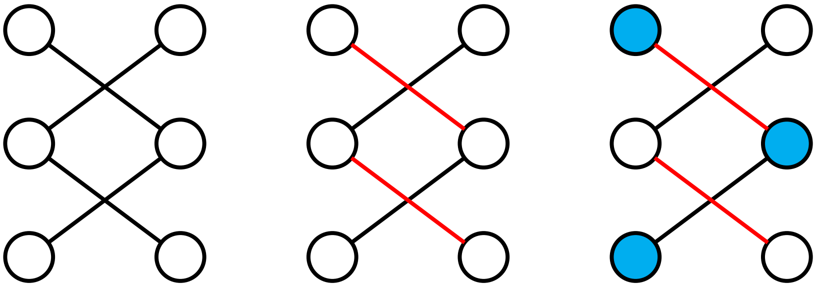
LP bound construction



LP bound construction

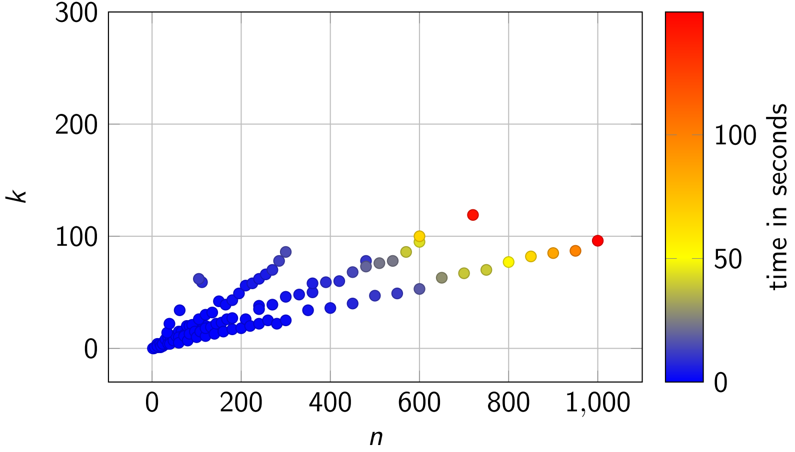


LP bound construction

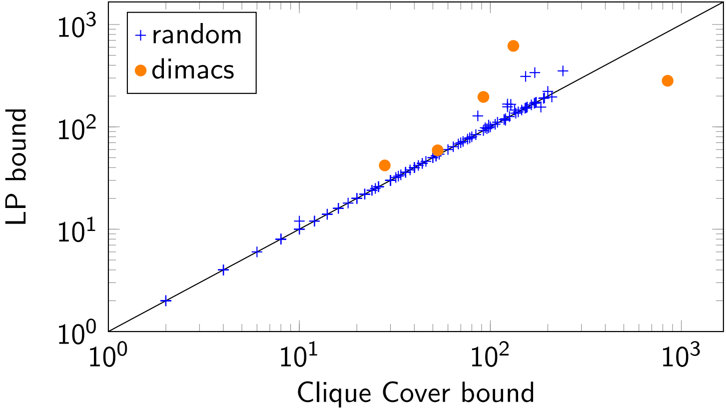


$$VC = (L \setminus Z) \cup (R \cap Z)$$

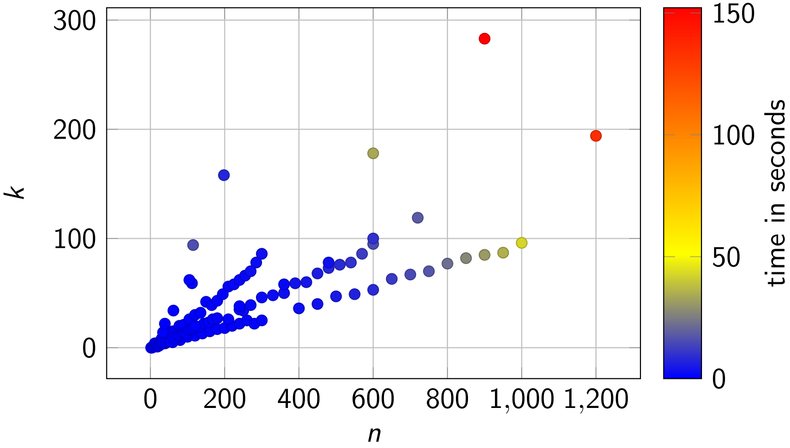
Linear Programming LB



Lower Bound Comparison



Combining Lower Bounds



Data Reduction

Degree 0 Rule

If there is an isolated vertex, remove it

Degree 1 Rule

If there is a degree-1 vertex, remove it and take its neighbour into the cover

High Degree Rule

If there is a vertex with degree greater than k, take it into the cover

Reduction Algorithm

procedure EXHAUSTIVE REDUCTION(G, k)
    applied $\leftarrow$ *false*
    repeat
        apply_deg_0(G, k)
        applied $\leftarrow$ *apply_deg_1*(G, k)
        applied $\leftarrow$ *apply_high_deg*(G, k)
    until not applied
    return G', k'

Runs in $\mathcal{O}(k \cdot n)$

Data Reduction

Properties of the reduced graph G'(V', E')?

- $\forall v \in V'.deg(v) \leq k'$
- $\forall v \in V'.deg(v) \geq 2$

Data Reduction

Properties of the reduced graph G'(V', E')?

- $\forall v \in V'.deg(v) \leq k' \leftarrow$ upper bound
- $\forall v \in V'.deg(v) \geq 2$

Limit size of our instance G' to k' !

Upper bound on graph size

Claim:
If $\Delta(G') \leq k'$ and $|E'| > k^2$ there is no VC of size k or smaller

Upper bound on graph size

Claim:
If $\Delta(G') \leq k'$ and $|E'| > k^2$ there is no VC of size k or smaller
Proof:
Let S be a vertex cover of G'.
Every $v \in S$ can cover at most k' edges. Furthermore $|S| \leq k$.
Then S can cover at most k^2 edges.

Upper bound on graph size

Claim:
If $|E'| \leq k'^2,$ then $|V'| \leq k'^2$

Upper bound on graph size

<u>Claim:</u>
If $|E'| \leq k'^2,$ then $|V'| \leq k'^2$
<u>Proof:</u>
By Degree-0 and Degree-1, G' has minimum degree at least 2.
By Handshake-Lemma $\sum_{v \in V'} deg(v) = 2|E'| \leq 2k'^2$
This implies $|V'| \leq k'^2$

Upper bound on graph size

<u>Claim:</u>
If $|E'| \leq k'^2,$ then $|V'| \leq k'^2$
<u>Proof:</u>
By Degree-0 and Degree-1, G' has minimum degree at least 2.
By Handshake-Lemma $\sum_{v \in V'} deg(v) = 2|E'| \leq 2k'^2$
This implies $|V'| \leq k'^2$

$\rightarrow$ reduced instance of G is bounded by k

To be specific: $|V'| + |E'| \subseteq \mathcal{O}(k'^2)$
$\Rightarrow$ Lower bounds become parameterised in k

Outlook on Future Improvements

- Updating matching in bipartite graph instead of recomputing
- Finish implementing existing reduction rules
- Developing more sophisticated reduction rules
  $\rightarrow$ to tighten the bound on the graph size
- Better branching rules
- Improve project modularity

Thanks for your attention

Questions?