Presentation 3

Fynn Lohren, Carsten Schubert, Leon Suchy

November 28, 2018

Structure

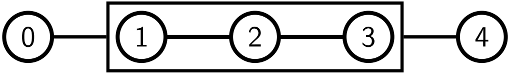
- Data Reductions
- Mirror branching
- Crown reduction
- Benchmarks
- Components

Data Reductions

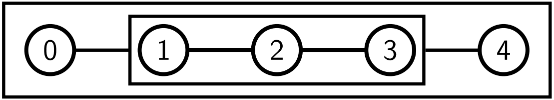
- Degree 0
- Degree 1
- Degree 2
- Degree Greater k
- Crown
- Dominate
- Unconfined

2-fold Undo - Wrong

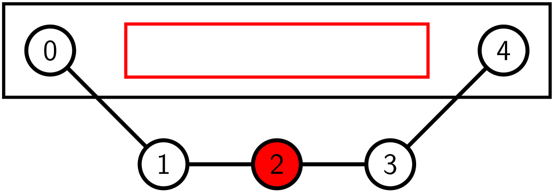
2-fold Undo - Wrong



2-fold Undo - Wrong

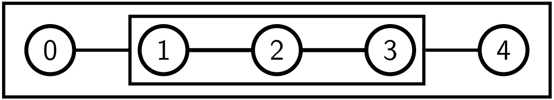


2-fold Undo - Wrong

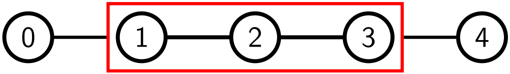


2-fold Undo - Wrong

2-fold Undo - Correct



2-fold Undo - Correct



2-fold Undo - Correct

Operation Stack

Undo order matters! $\rightarrow$ Stack of operations

group 7	5
group 1	4
take 2	3
group 3	2
take 5	1
invalidate 0	0

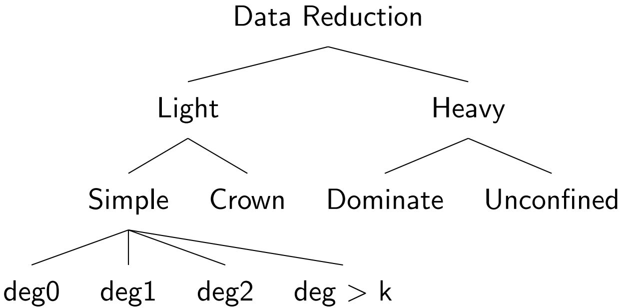
$\Rightarrow 5, 4, 3, 2, 1, 0$

Reductions push their operations onto the stack

Configs

- 8 different reductions
  $\rightarrow$ Many possible combinations to benchmark
- Enable/Disable
- Script can batch execute them
- Order of application
- Hierarchy
  $\rightarrow$ Apply a group of reductions before another $\rightarrow$ Crown reduction

Hierarchy

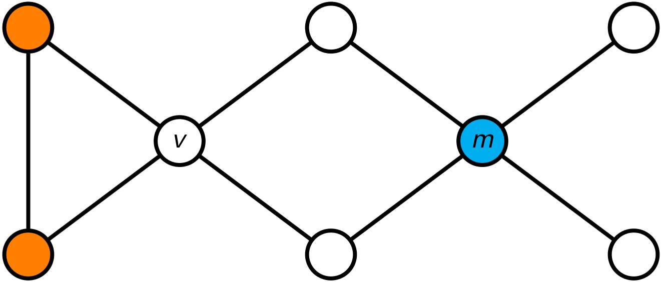

Data Reduction
Light
Heavy
Simple
Crown
Dominate
Unconfined
deg0
deg1
deg2
deg > k

Mirror branching

Mirror branching is a strategy to enhance the standard Max Degree Branching that we use in our implementation.

Original source is the following paper:
"A Measure & Conquer Approach for the Analysis of Exact Algorithms" by Fedor V. Fomin, Fabrizio Grandoni and Dieter Kratsch

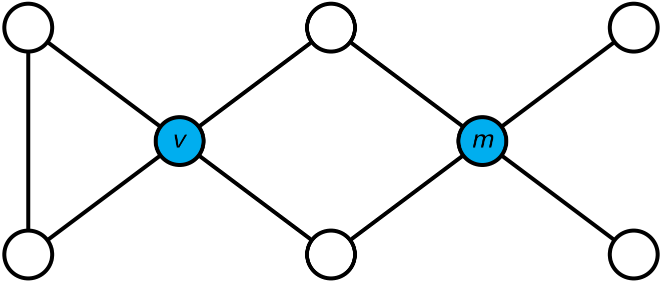
Mirror branching



m is called *mirror* of v if:

- $m \in N^2(v)$
- $N(v) \setminus N(m)$ induces a clique

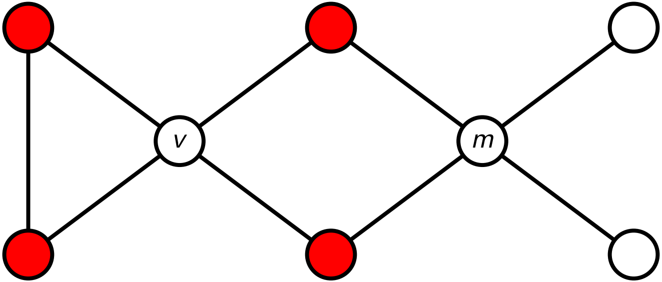
Mirror branching



Refined branching rule:

- Either we take v with all its mirrors into the MVC

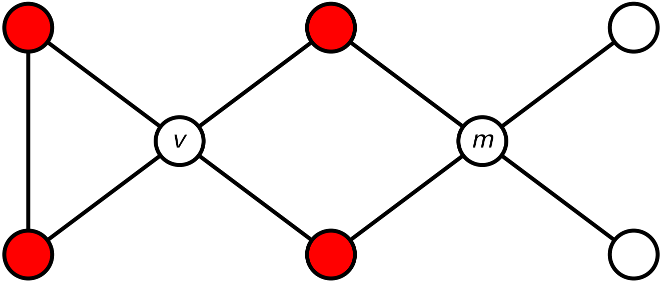
Mirror branching



Refined branching rule:

- Either we take v with all its mirrors into the MVC
- Or we take $N(v)$ into the MVC

Mirror branching



Refined branching rule:

- Either we take v with all its mirrors into the MVC
- Or we take $N(v)$ into the MVC

$\Rightarrow$ Always scan for mirrors before branching on a vertex!

Mirror branching enhancement

Previously, we built a branching tree from v with max size of

$$T(n-1) + T(n - |N[v]|)$$

Mirror branching enhancement

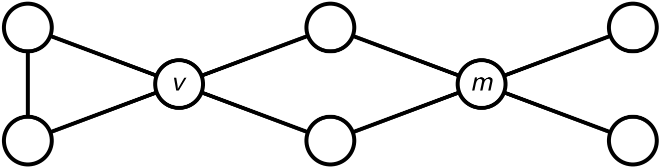
Previously, we built a branching tree from v with max size of

$$T(n-1) + T(n - |N[v]|)$$

Obviously the search tree is unbalanced.
With mirrors, we attend this issue:

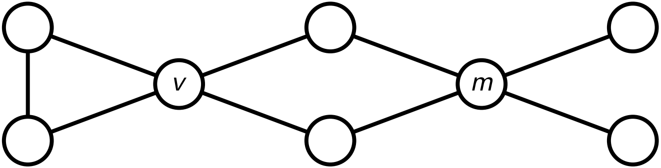
$$T(n - |\mathcal{M}[v]|) + T(n - |N[v]|)$$

Mirror branching correctness



Assume we branch on v and v has a mirror m.

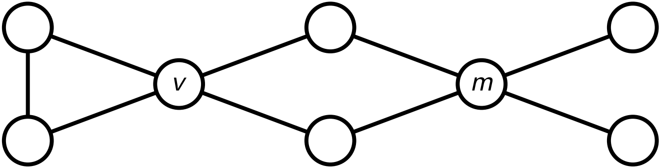
Mirror branching correctness



Assume we branch on v and v has a mirror m.

$\rightarrow$ Then we do not want to include $N(v) \cap N(m)$ in the MVC.

Mirror branching correctness

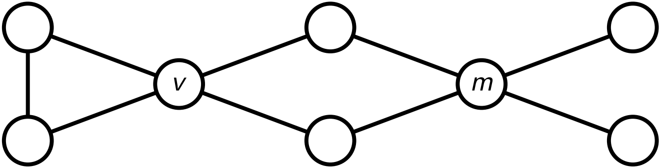


Assume we branch on v and v has a mirror m.

$\rightarrow$ Then we do not want to include $N(v) \cap N(m)$ in the MVC.

$\Rightarrow$ Because if we did, we might as well just discard v and cover the clique with its remaining neighbours!

Mirror branching correctness



Assume we branch on v and v has a mirror m.

$\rightarrow$ Then we do not want to include $N(v) \cap N(m)$ in the MVC.

$\Rightarrow$ Because if we did, we might as well just discard v and cover the clique with its remaining neighbours!

Thus, we have to include m in the MVC
in order to cover its edges to $N(v) \cap N(m)$.

Mirror branching runtime and further heuristics

Our implementation finds mirrors of v in $O(\delta(v)^3) \leq O(k^3)$, however it makes use of dynamic programming to remember already encountered edges.

Mirror branching runtime and further heuristics

Our implementation finds mirrors of v in $O(\delta(v)^3) \leq O(k^3)$,
however it makes use of dynamic programming
to remember already encountered edges.

Further branching improvement:
From the max degree vertices, we select the vertex v that
minimizes $E(N(v))$.

Mirror branching runtime and further heuristics

Our implementation finds mirrors of v in $O(\delta(v)^3) \leq O(k^3)$,
however it makes use of dynamic programming
to remember already encountered edges.

Further branching improvement:
From the max degree vertices, we select the vertex v that minimizes $E(N(v))$.

$\rightarrow$ increases the chance to find mirrors

$\rightarrow$ reduces mirror searching time

Crown Reduction

Finding crowns is done by computing a maximal matching
$\rightarrow$ non matched vertices form an IS
Build a bipartite graph with V($IS \cup N(IS)$, E)

Crown Reduction

Finding crowns is done by computing a maximal matching
$\rightarrow$ non matched vertices form an IS
Build a bipartite graph with V($IS \cup N(IS)$, E)

Observation:
size of IS depends on the maximal matching
$\rightarrow$ minimum maximal matching is edge dominating set

Crown Reduction

Finding crowns is done by computing a maximal matching
$\rightarrow$ non matched vertices form an IS
Build a bipartite graph with V($IS \cup N(IS)$, E)

Observation:
size of IS depends on the maximal matching
$\rightarrow$ minimum maximal matching is edge dominating set
$\rightarrow$ our approach: randomize the matching!

Crown Reduction

Runtime:

$\mathcal{O}(m \cdot \sqrt{n})$

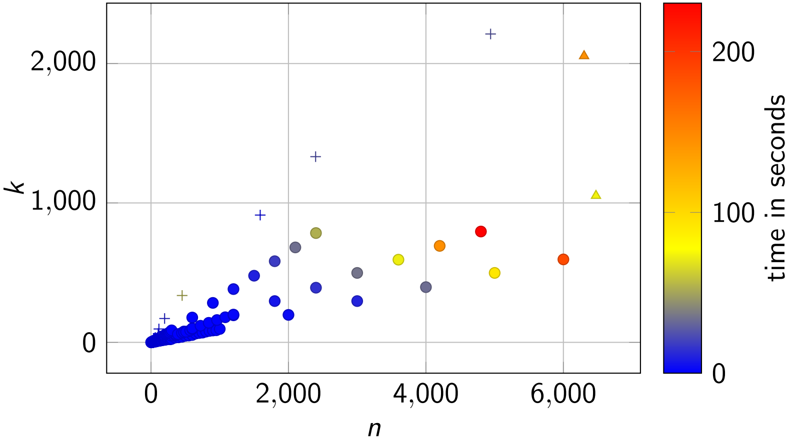
Note:

n $\rightarrow |IS| + |N(IS)|$

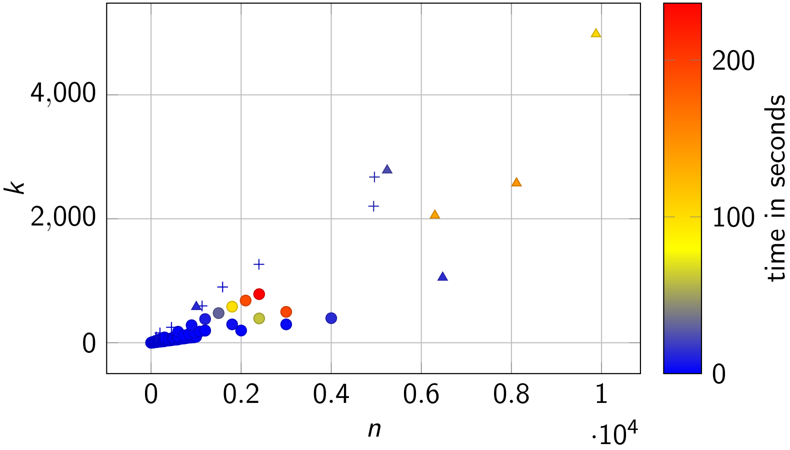
m $\rightarrow E(IS \cup N(IS))$

$\rightarrow$ no guarantee on runtime

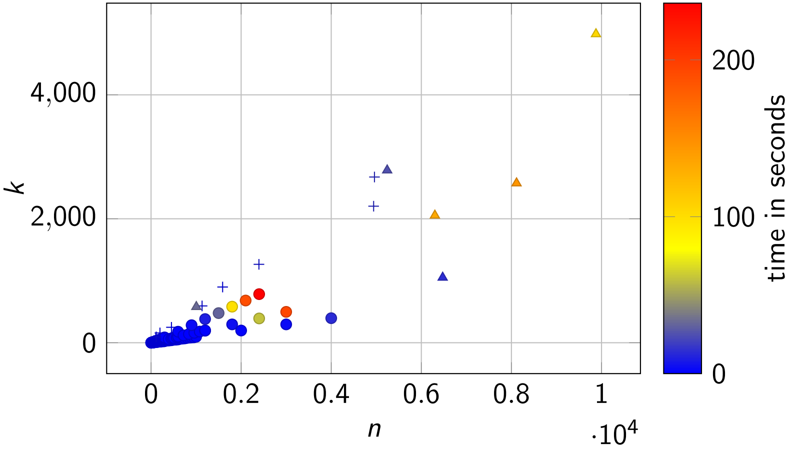
0,1



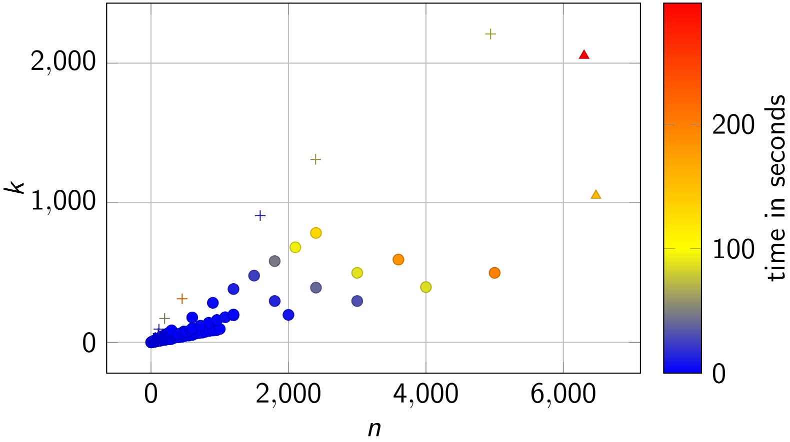
0,1,2,k,dominate



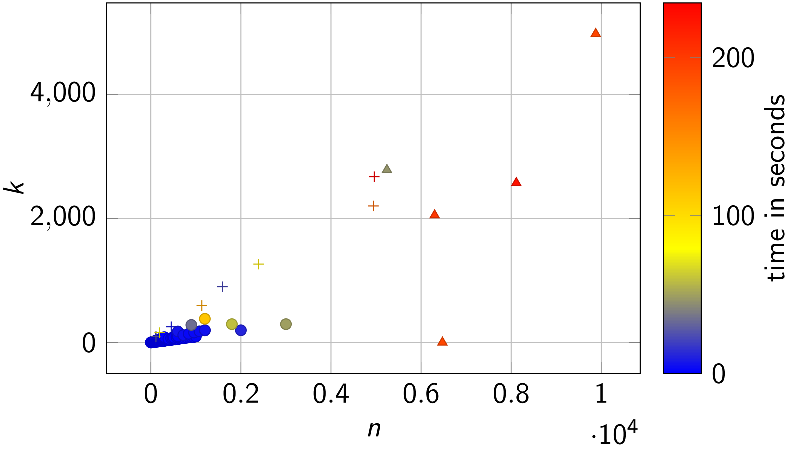
0,1,2,k,unconfined



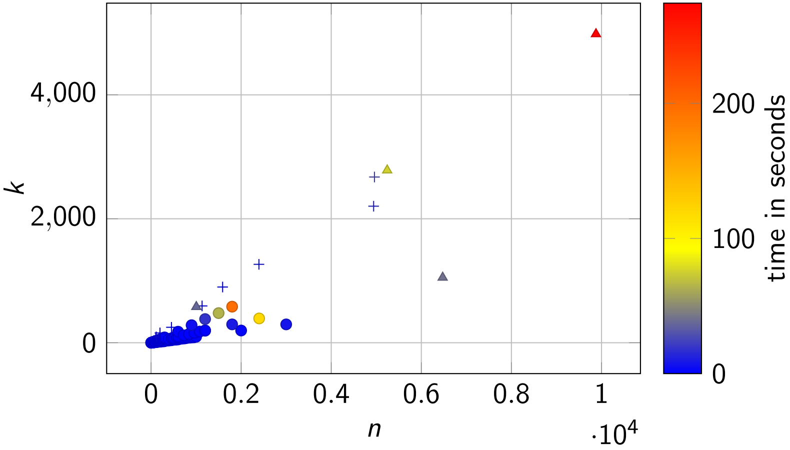
0,1,2,k,dominate, unconfined



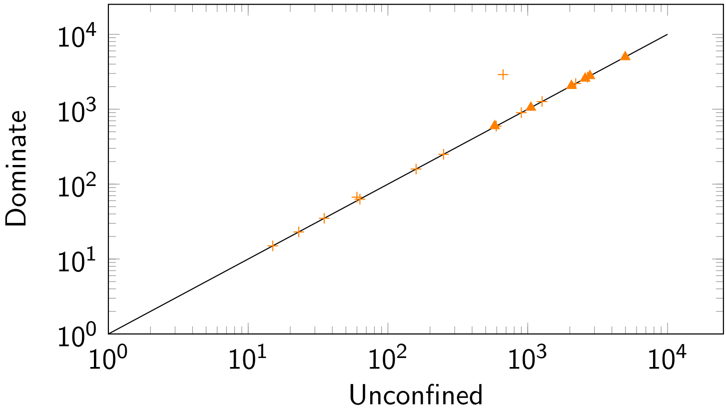
0,1,2,k,crown



All Reductions

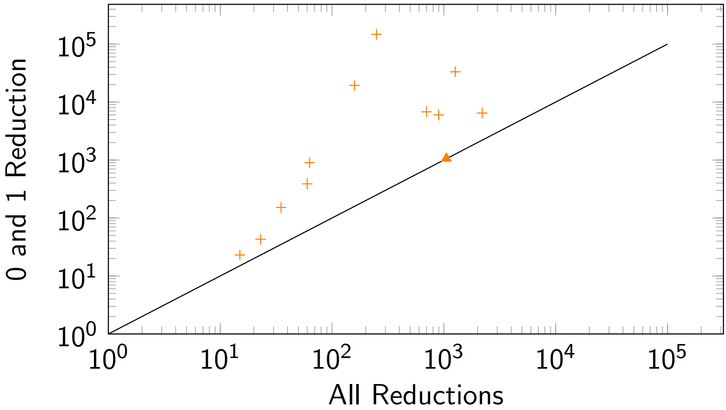


Unconfined and Dominate Comparison



Only instances solved by both variants are listed

0 and 1 to All Reductions comparison



Only instances solved by both variants are listed

Components

1. Use DFS to find components
2. Create subgraphs for each component
3. Sort them to process easy ones first
4. Solve on each subgraph
5. Combine the found solutions

Faster because we break earlier and reduce k early.

Outlook on Future Improvements

- Performance Improvements for Reductions
- LP reduction via Flow
- Degree 3 reduction
- Refactor Solver
- Priority Queue for Vertex Selection

Questions?