Heuristics for Twin-Width

Denis Koshelev

Technische Universität Berlin

December 12, 2023

Agenda

1. Introduction to Twin-Width problem

Agenda

1. Introduction to Twin-Width problem
2. Heuristics
   - Intra-Set Pairing
   - Random Walk
   - Combination of both

Agenda

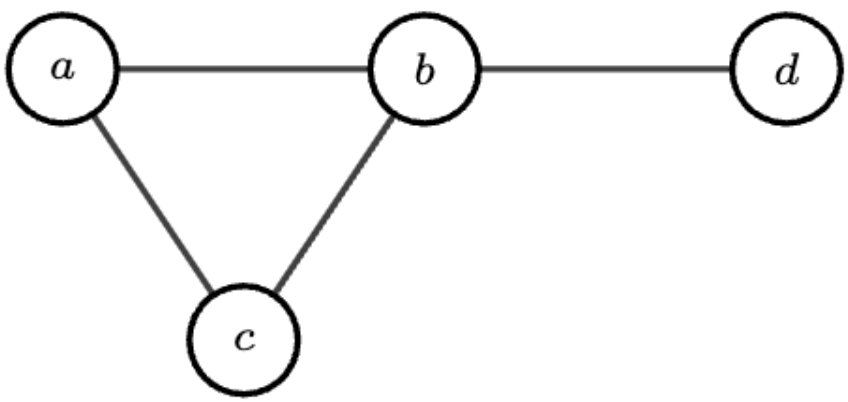
1. Introduction to Twin-Width problem
2. Heuristics
   - Intra-Set Pairing
   - Random Walk
   - Combination of both
3. Score functions and threshold choice

Agenda

1. Introduction to Twin-Width problem
2. Heuristics
   - Intra-Set Pairing
   - Random Walk
   - Combination of both
3. Score functions and threshold choice
4. Optimizations
   - Almost reduction rules
   - Caching mechanism
5. Conclusion

Introduction to Twin-Width problem

The goal is to collapse the graph into a single vertex by **merging vertices**:

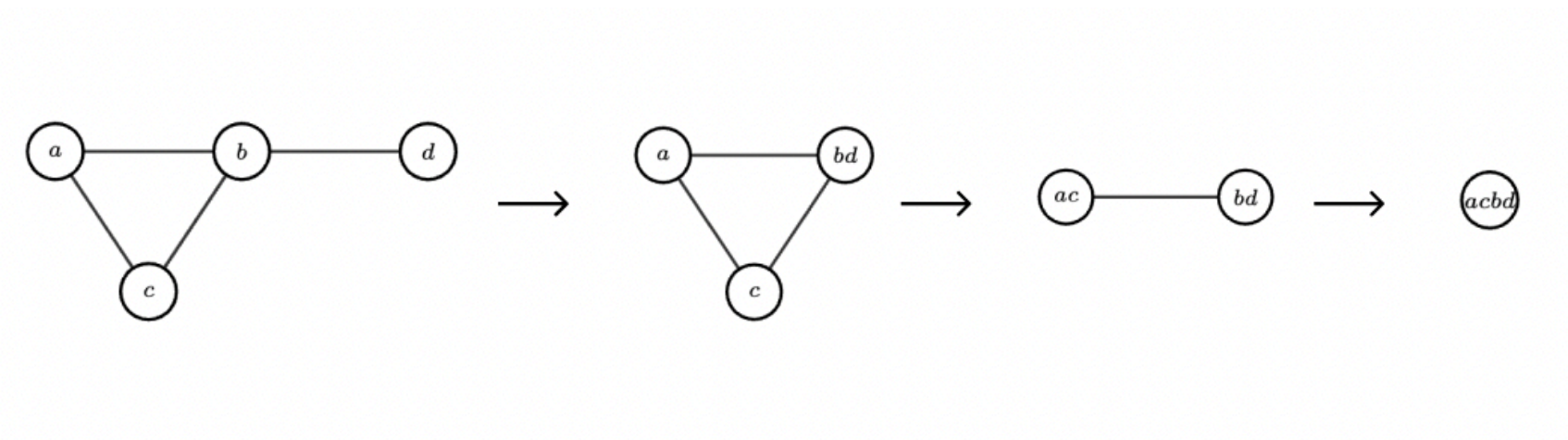


Introduction to Twin-Width problem

To define order of merge operations we use a **contraction sequence**

For example, $S = ((b,d), (a, c), (ac, bd))$

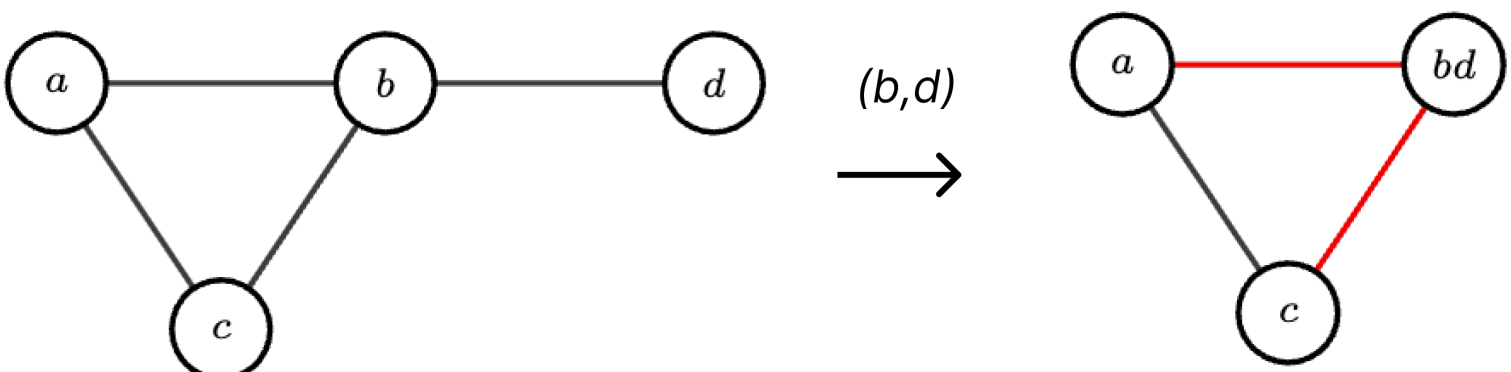
Illustrated by:



Introduction to Twin-Width problem

After merging, we mark every unique edge of the merged vertices red

$N(b) = \{a, c, d\}, N(d) = \{b\}$ and $N(b) \triangle N(d) = \{a, c\}$

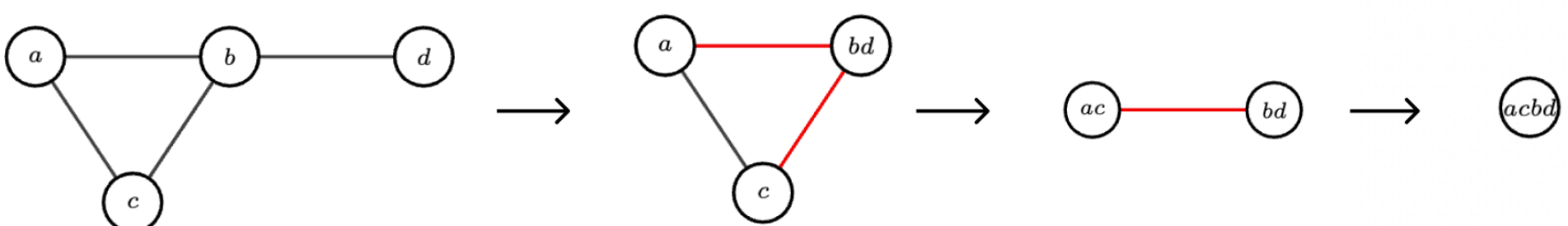


Introduction to Twin-Width problem

We have the same contraction sequence $S = ((b,d), (a, c), (ac, bd))$

But now we can measure its width, where G_i is a graph after i steps:

$$\text{width}(S) = \max_i \left\{ \max_{v \in V(G_i)} \text{red}(v) \right\} = 2$$



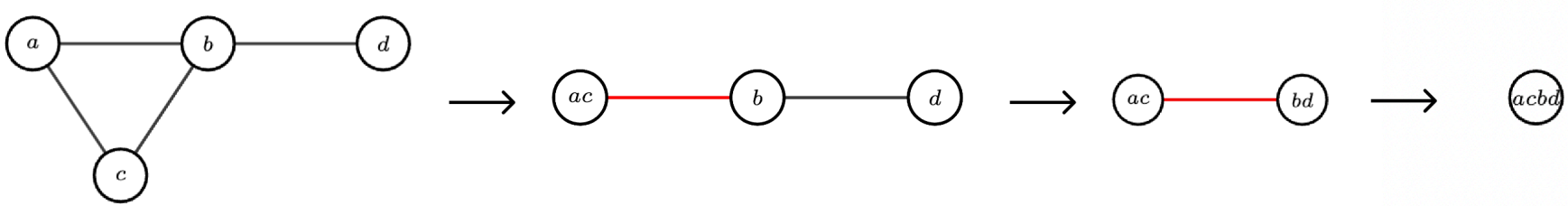
Introduction to Twin-Width problem

The **twin-width** of the graph *tww(G)* represents the *minimum possible width* achievable through a contraction sequence

Introduction to Twin-Width problem

The **twin-width** of the graph *tww(G)* represents the *minimum possible width* achievable through a contraction sequence

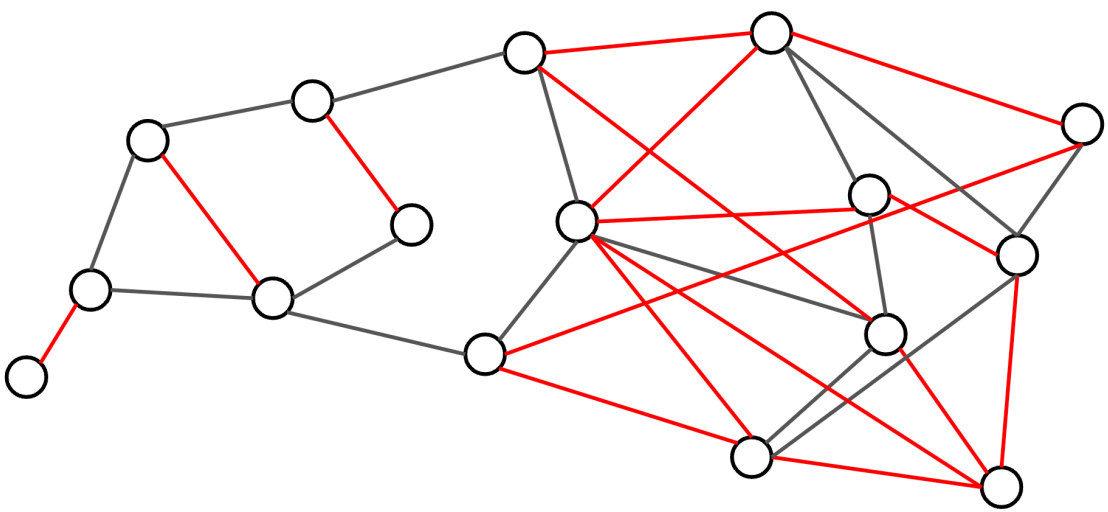
An optimal contraction sequence $S^* = ((a, c), (b, d), (ac, bd))$ that results in $\mathrm{tww}(G) = \mathrm{width}(S^*) = 1$



How do we find good vertex pairs that minimize number of red edges?

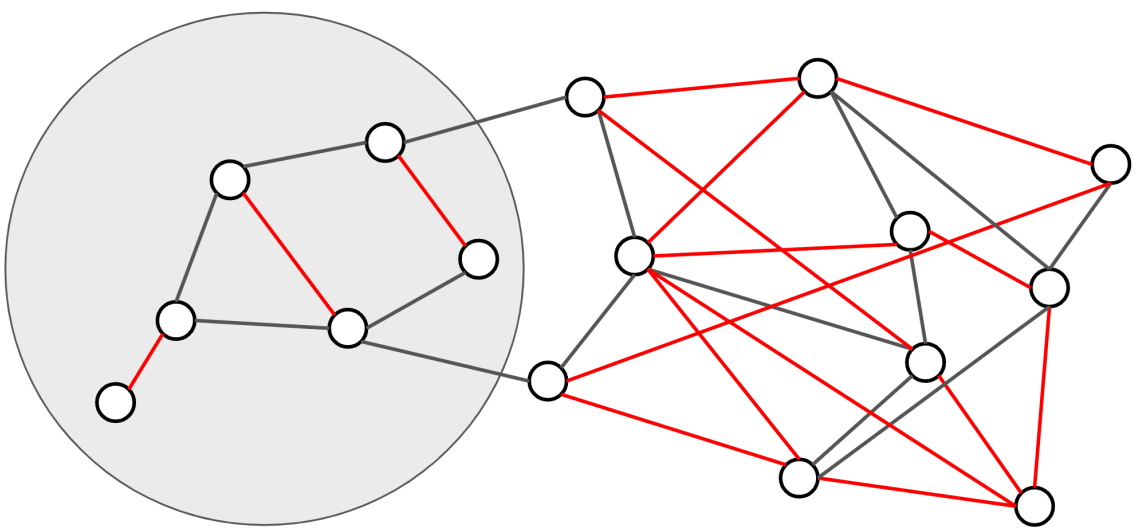
Heuristics: Intra-Set Heuristic

Suppose we have a following graph after some contraction steps:



Heuristics: Intra-Set Heuristic

Find the vertices with minimum (red) degree and contract them first:



Heuristics: Intra-Set Heuristic

Algorithm:

1. Pick η vertices with the lowest **degree** *or* **red degree**

Heuristics: Intra-Set Heuristic

Algorithm:

1. Pick η vertices with the lowest **degree** *or* **red degree**
2. Among this vertex set, compute *score* for every possible pair

Heuristics: Intra-Set Heuristic

Algorithm:

1. Pick η vertices with the lowest **degree** *or* **red degree**
2. Among this vertex set, compute *score* for every possible pair
3. Contract pair with the lowest score

Heuristics: Intra-Set Heuristic

Algorithm:

1. Pick η vertices with the lowest **degree** *or* **red degree**
2. Among this vertex set, compute *score* for every possible pair
3. Contract pair with the lowest score
4. Repeat until we end up with a single vertex.

Heuristics: Intra-Set Heuristic

Algorithm:

1. Pick η vertices with the lowest **degree** *or* **red degree**
2. Among this vertex set, compute *score* for every possible pair
3. Contract pair with the lowest score
4. Repeat until we end up with a single vertex.

Score is a size of **symmetric difference** of vertices' neighborhoods:

$$\text{score}(v, u) := |N(v) \triangle N(u)| = |N(v) \cup N(u) - \ N(v) \cap N(u)|$$

Heuristics: Intra-Set Heuristic

Intuition behind:

1. **Minimum degree**: while collapsing the graph vertices with the smallest degree most likely would cause fewer red edges

Heuristics: Intra-Set Heuristic

Intuition behind:

1. **Minimum degree**: while collapsing the graph vertices with the smallest degree most likely would cause fewer red edges
2. **Minimum red degree**: contracting vertices with large red degree might increase width of the contraction sequence after merge

Heuristics: Intra-Set Heuristic

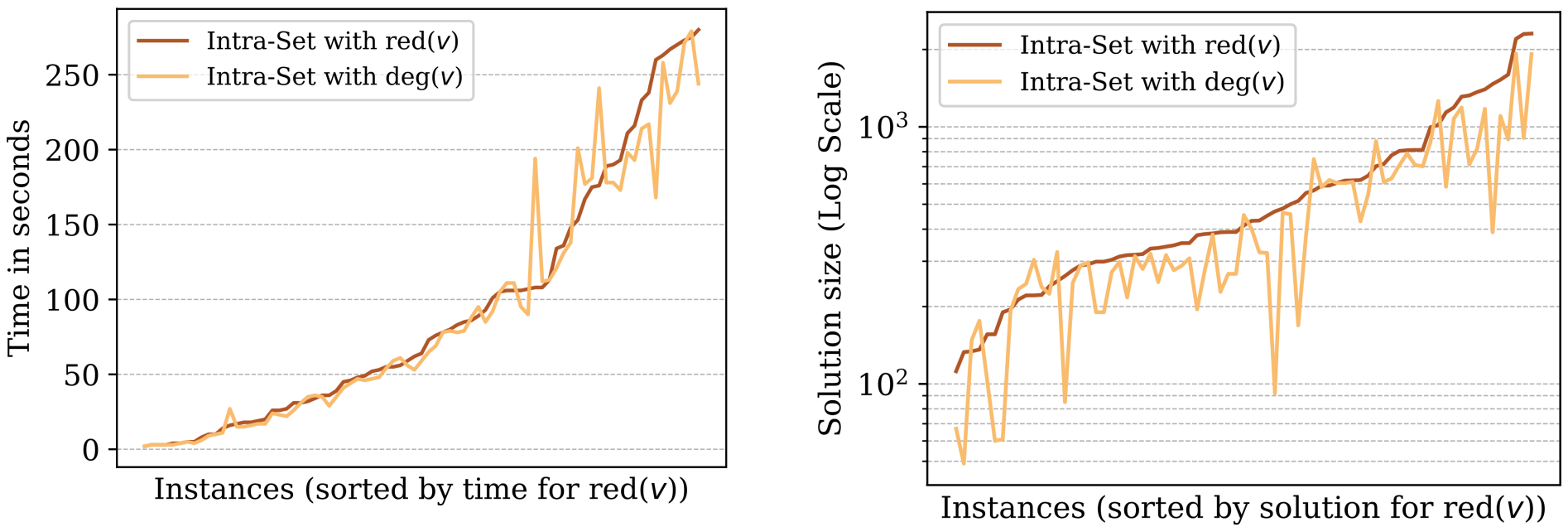


Figure 1: Comparison of intra-set heuristics based on low (red) degree

Intra-Set heuristic based on **vertex degree** prevails. Why?

Heuristics: Intra-Set Heuristic

Intra-Set heuristic based on **vertex degree** prevails. Why?

Red degree based heuristic doesn't consider for neighborhood sizes
$\rightarrow$ At certain iteration step might not find a good pair with low score
$\rightarrow$ Since neighborhood size isn't minimized, width increase is large

Heuristics: Random Walk Heuristic

Observation 1: Two vertices v and u can have common neighbors if and only if they are within the second neighborhood of each other

Heuristics: Random Walk Heuristic

Observation 1: Two vertices v and u can have common neighbors if and only if they are within the second neighborhood of each other

Figure 3: Illustrations of direct and two-hop neighbors in a graph

Heuristics: Random Walk Heuristic

Observation 1: Two vertices v and u can have common neighbors if and only if they are within the second neighborhood of each other

Heuristics: Random Walk Heuristic

Observation 1: Two vertices v and u can have common neighbors if and only if they are within the second neighborhood of each other

Random Walk heuristic:

1. Pick η vertices with the lowest **degree** or **red degree**

Heuristics: Random Walk Heuristic

Observation 1: Two vertices v and u can have common neighbors if and only if they are within the second neighborhood of each other

Random Walk heuristic:

1. Pick η vertices with the lowest **degree** or **red degree**
2. For each, perform a random walk with 1 or 2 steps

Heuristics: Random Walk Heuristic

Observation 1: Two vertices v and u can have common neighbors if and only if they are within the second neighborhood of each other

Random Walk heuristic:

1. Pick η vertices with the lowest **degree** or **red degree**
2. For each, perform a random walk with 1 or 2 steps
3. Compute score for such each pair

Heuristics: Random Walk Heuristic

Observation 1: Two vertices v and u can have common neighbors if and only if they are within the second neighborhood of each other

Random Walk heuristic:

1. Pick η vertices with the lowest **degree** or **red degree**
2. For each, perform a random walk with 1 or 2 steps
3. Compute score for such each pair
4. Contract pair with the lowest score

Heuristics: Random Walk Heuristic

Observation 1: Two vertices v and u can have common neighbors if and only if they are within the second neighborhood of each other

Random Walk heuristic:

1. Pick η vertices with the lowest **degree** or **red degree**
2. For each, perform a random walk with 1 or 2 steps
3. Compute score for such each pair
4. Contract pair with the lowest score
5. Repeat until we end up with a single vertex.

Heuristics: Random Walk Heuristic

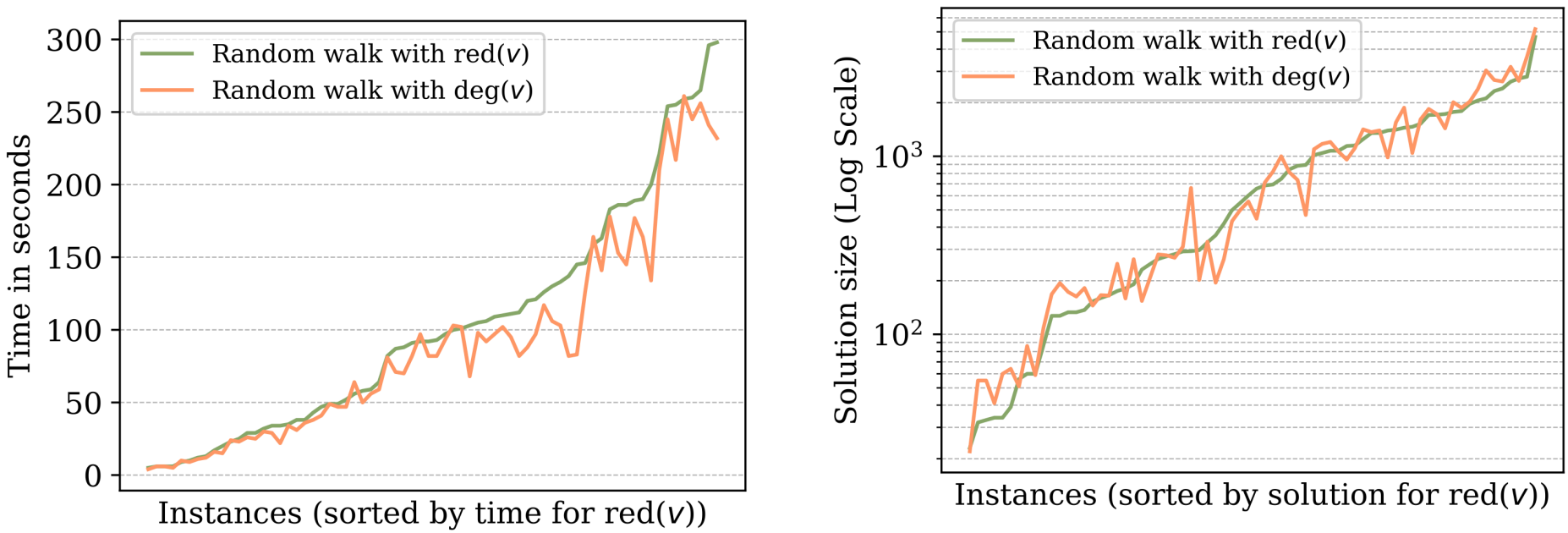


Figure 4: Comparison of random walks based on low (red) degree

Let’s compare best of both versions of heuristics with each other.

Heuristics: Combine both

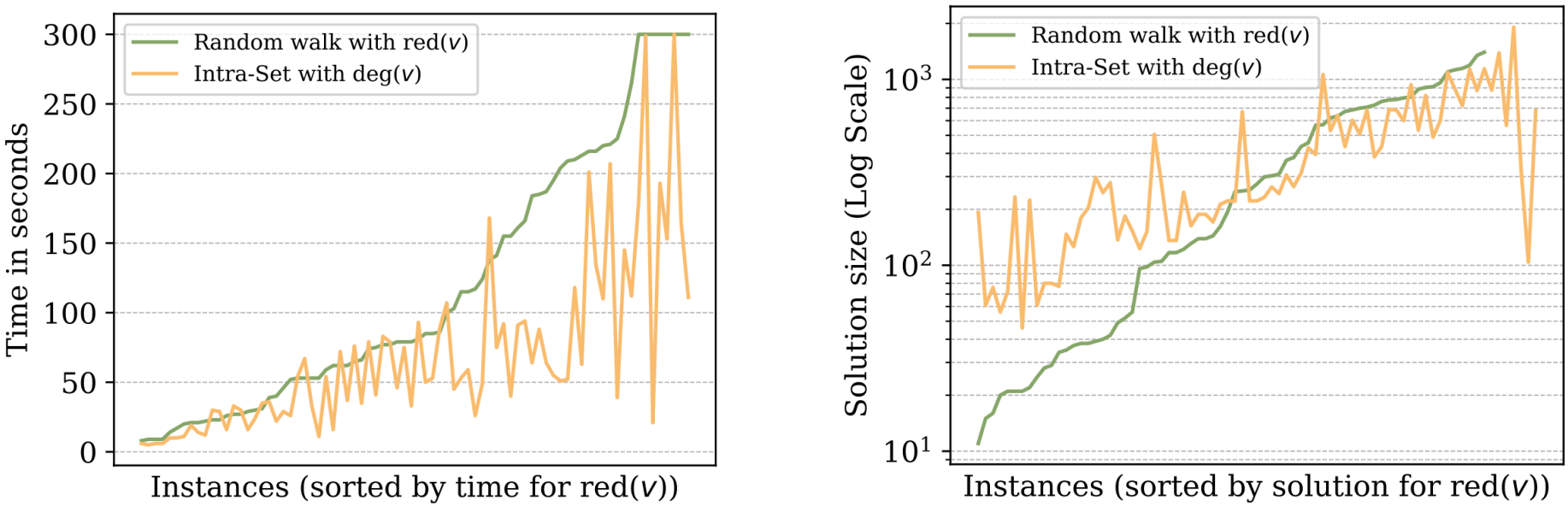


Figure 5: Comparison of best versions of both heuristics

Heuristics: Combine both

An interesting trend emerged: random walk heuristic outperformed intra-set on the graphs with low average degree distribution.

	Instances	V	E	Mean Degree	Avg Deviation	Intra-Set	Random Walk
Intra-Set	heuristic_006.gr	1500	5.69×10^5	758	193	**871**	1346
	heuristic_007.gr	1500	8.47×10^5	1129	96	**601**	960
	heuristic_013.gr	4087	1.85×10^5	90	57	**312**	435
	heuristic_017.gr	9072	4.65×10^5	102	107	**722**	1146
Random walk	heuristic_029.gr	14822	3.5×10^5	155	3	56	**20**
	heuristic_033.gr	18454	1.17×10^5	13	3	224	**22**
	heuristic_036.gr	20082	1.31×10^5	13	2	233	**21**
	heuristic_057.gr	32766	1.60×10^5	9	3	67	**11**

Heuristics: Combine both

If the **average degree deviation** is large, the random walk tends to consider pairs with large neighborhoods, resulting in a **distruption of our vertex selection logic**

$\rightarrow$ Contraction width increases drastically at certain point

Heuristics: Combine both

If the **average degree deviation** is large, the random walk tends to consider pairs with large neighborhoods, resulting in a **distruption of our vertex selection logic**

$\rightarrow$ Contraction width increases drastically at certain point

Heuristic choice rule: if average degree deviation is less than 25, random walk is employed; otherwise, the intra-set heuristic is used.

Heuristics: Random Walk Heuristic

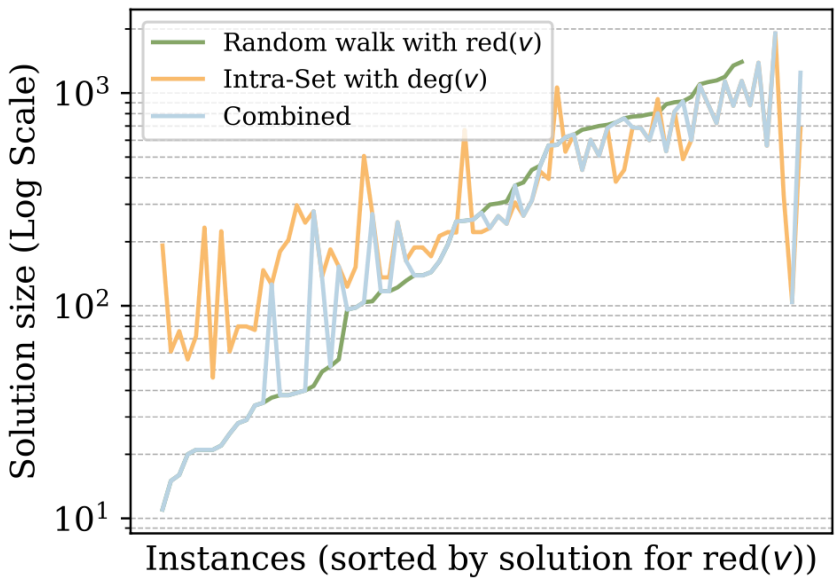


Figure 6: Comparison of previous heuristics with combined approach

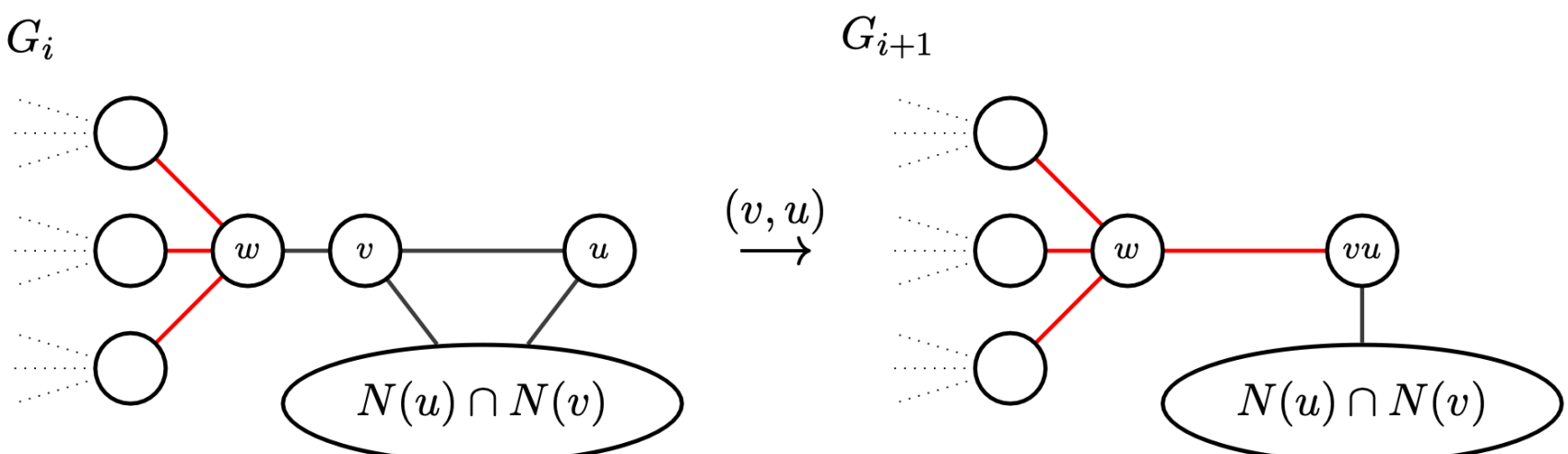
Score functions

As mentioned before, our main score function for vertex pair is the following:

$$\text{score}(v, u) := |N(v) \cup N(u) - \ N(v) \cap N(u)|$$

Score functions

However, it accounts only locally for two vertices, even though merge might case width increase indirectly:



Score functions

We try different variations similar to our main function. Second approach considers **only red edges emerged** at the current iteration:

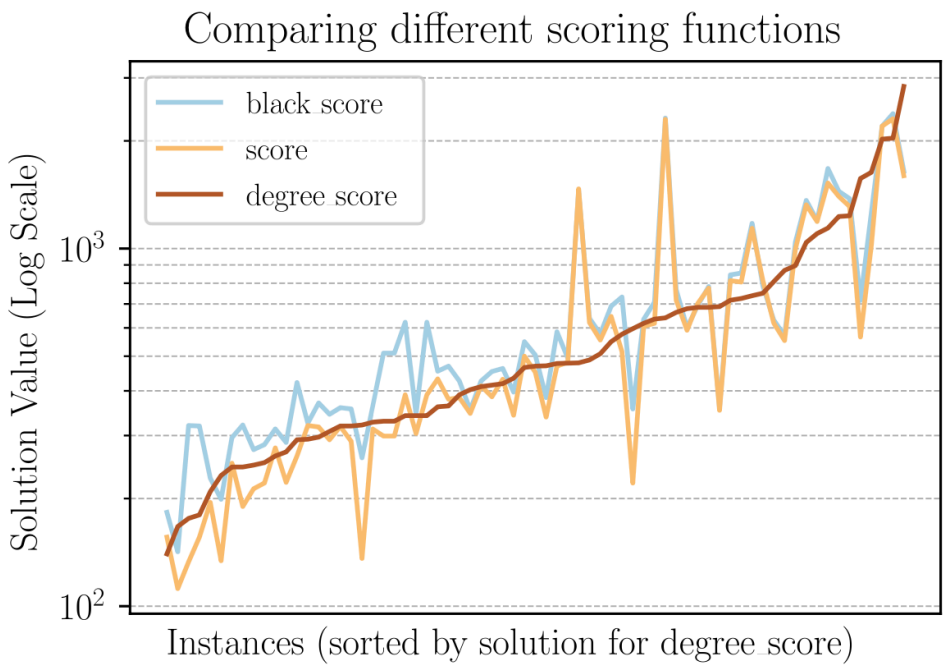
$$\text{black\_score}(v, u) := |N_B(v) \cup N_B(u) - \ N_B(v) \cap N_B(u)|$$

Score functions

Third approach additionally to new red edges considers neighborhood **cardinality difference** and **penalizes for existing red edges**:

$$\text{degree\_score}(v,u) := \text{black\_score}(v,u) + \underbrace{|\deg(v) - \deg(u)|}_{\substack{\text{Makes sure vertices} \\ \text{have similar degree}}} + \underbrace{\text{red}(v) + \text{red}(u)}_{\text{penalizes for existing red edges}}$$

Score functions



Metric	score	degree_score
Best solved	**36**	33
μ_{geo}	**467**	490
μ	614	**611**
σ	510	**487**

Table 4.2: Performance metrics for score function. Better values are bolded.

Figure 4.9: Analysis of scoring functions: the plot shows a difference in solution size on a logarithmic scale.

Optimizations

1. **Almost reduction rules**
   - Twin rule: eliminate all twins before employing the heuristic
   - One Degree rule: try to contract **some** of the one degree vertices in a randomized manner

Optimizations

1. **Almost reduction rules**
   - Twin rule: eliminate all twins before employing the heuristic
   - One Degree rule: try to contract **some** of the one degree vertices in a randomized manner

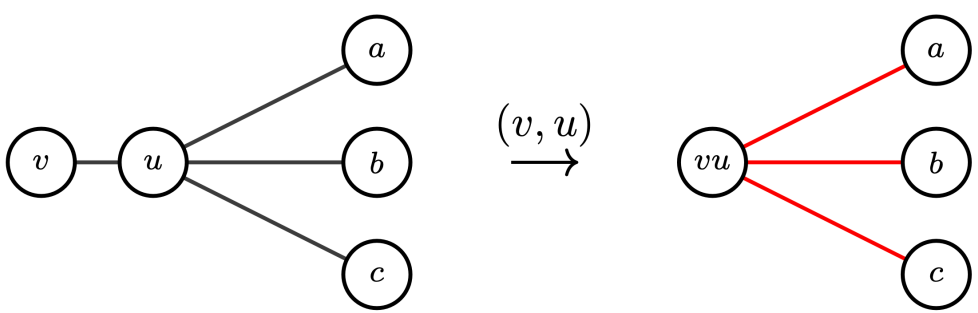


Figure 8: Problem with one degree vertices

Optimizations

2. **Threshold choice**
   - How to choose η for *candidates* for both heuristics?
   - After experimenting we decide to use $\eta = 3 \cdot \lceil \log(V) \rceil$

Optimizations

2. **Threshold choice**
   - How to choose η for *candidates* for both heuristics?
   - After experimenting we decide to use $\eta = 3 \cdot \lceil \log(V) \rceil$
3. **Density trick**
   - By dense graphs, we solve Twin-Width on its complement

Optimizations

2. **Threshold choice**
   - How to choose η for *candidates* for both heuristics?
   - After experimenting we decide to use $\eta = 3 \cdot \lceil \log(V) \rceil$
3. **Density trick**
   - By dense graphs, we solve Twin-Width on its complement
4. **Caching scores mechanism**
   - Instead of recomputing scores in every iteration, we introduce caching table
   - Clear table with scores every 100 iteration steps

Conclusion

1. **Combined approach** turned out to be the most efficient one

Conclusion

1. **Combined approach** turned out to be the most efficient one
2. Currently, there is **no generic approach**: most solvers, including the one presented, use mix of different heuristics

Conclusion

1. **Combined approach** turned out to be the most efficient one
2. Currently, there is **no generic approach**: most solvers, including the one presented, use mix of different heuristics
3. Little optimizations tricks **reduce computation time** but **don't significantly improve** the solution size.

Conclusion

1. **Combined approach** turned out to be the most efficient one
2. Currently, there is **no generic approach**: most solvers, including the one presented, use mix of different heuristics
3. Little optimizations tricks **reduce computation time** but **don't significantly improve** the solution size.

Future steps:

1. Other non-trivial reduction rules

Conclusion

1. **Combined approach** turned out to be the most efficient one
2. Currently, there is **no generic approach**: most solvers, including the one presented, use mix of different heuristics
3. Little optimizations tricks **reduce computation time** but **don't significantly improve** the solution size.

Future steps:

1. Other non-trivial reduction rules
2. New vertex scoring methods using Jaccard similarity coefficient

Conclusion

1. **Combined approach** turned out to be the most efficient one
2. Currently, there is **no generic approach**: most solvers, including the one presented, use mix of different heuristics
3. Little optimizations tricks **reduce computation time** but **don't significantly improve** the solution size.

Future steps:

1. Other non-trivial reduction rules
2. New vertex scoring methods using Jaccard similarity coefficient
3. Explore changes of computed scores after multiple iterations

Thank you for your attention. Questions?