State of the Art: Dominating Set Problem

(https://epubs.siam.org/doi/pdf/10.1137/1.9781611976700.38)

1. Problem Description

The Dominating Set Problem is a fundamental problem in graph theory and combinatorial optimization. Given a graph G = (V, E), a dominating set D that is a subset of V is a set of vertices such that every node in the graph is either in D or adjacent to at least one node in D. This problem has applications in various domains, including:

- Network design and communication (sensor placement, routing in wireless networks)
- Social network analysis (influencer selection, recommendation systems)
- Computational biology (gene regulatory networks, protein interaction networks)

Due to its NP-hard nature, finding an exact solution is computationally infeasible for large graphs, leading to the development of heuristic, approximation, and deep learning-based approaches.

2. Existing Approaches for the Problem

Over the years, different methods have been proposed to solve the Dominating Set problem. These can be broadly categorized as follows:

2.1 Traditional Algorithmic Approaches

Exact Methods:

- **Integer Linear Programming (ILP)**: Models the problem as an ILP and uses solvers like CPLEX or Gurobi, LINK
- **Branch-and-Bound and Branch-and-Cut Algorithms**: Used for small graph instances but scale poorly with large graphs.

Approximation Algorithms:

- **Greedy Algorithm**: Iteratively selects the node that covers the most uncovered nodes.
- **LP Rounding Approaches**: Uses linear programming relaxation followed by rounding strategies.
- **Primal-Dual Algorithms**: Provide improved theoretical guarantees.

2.2 Heuristic and Metaheuristic Approaches

- **Genetic Algorithms (GA)**: Uses evolutionary principles to optimize the selection of nodes in the dominating set
- **Simulated Annealing (SA)**: Explores the search space while avoiding local optima, LINK
- **Ant Colony Optimization (ACO)**: Models the problem as a path-finding process using artificial ants, LINK

2.3 Graph Neural Network (GNN)-Based Approaches

With the rise of deep learning, Graph Neural Networks (GNNs) have been explored for solving combinatorial optimization problems like MDS. Key developments include:

- **Graph Convolutional Networks (GCNs)** [Kipf & Welling, 2017] – Effective for node classification but struggles with expressiveness for combinatorial problems.
- **Graph Isomorphism Networks (GINs)** [Xu et al., 2019] – Stronger expressiveness but limited in approximation quality.
- **Coloring-based GNNs (CPNGNNs)** [Sato et al., 2019] – Introduces weak 2-coloring to improve representation.
- **Random Feature GNNs (rGINs)** [Sato et al., 2021] – Uses random node features to enable near-optimal approximation of MDS.

3. Benchmark Instances

Link: https://www.ic.unicamp.br/~cid/Problem-instances/Eternal-Dominating-Set/

To evaluate the performance of different approaches, the following benchmark datasets are commonly used:

3.1 Synthetic Datasets

- **Random-regular Graphs**: Graphs with a fixed degree for each node.
- **Barabasi-Albert (BA) Graphs**: Scale-free networks used to model real-world phenomena.
- **Erdos-Renyi (ER) Graphs**: Random graphs where edges appear with a fixed probability.

3.2 Real-World Datasets

- **Social Networks**: Facebook, Twitter, and citation networks.

- **Biological Networks**: Protein interaction graphs.
- **Infrastructure Networks**: Road and power grid networks.

3.3 Our Solution - Dominating Set

- ***Dataset: https://github.com/MarioGrobler/ds_verifier***

Example:

bremen_subgraph_50.gr

PAIRS: 1-7, 1-10, 1-58, 2-3, 2-35, 3-11, 3-35, 4-45, 4-5, 4-32, 4-36, 5-46, 5-33, 5-18, 6-46, 6-7, 6-29, 7-19, 7-33, 8-12, 8-9, 8-47, 8-55, 9-13, 9-31, 9-56, 10-33, 10-57, 10-20, 11-12, 11-53, 11-30, 12-54, 12-60, 13-14, 13-48, 13-59, 14-56, 14-15, 15-51, 16-19, 16-42, 17-50, 17-49, 17-62, 18-60, 18-36, 20-60, 20-57, 20-21, 21-34, 21-30, 22-45, 22-28, 22-23, 22-26, 23-24, 23-26, 24-25, 24-61, 25-43, 26-41, 27-49, 27-31, 27-32, 28-29, 30-40, 31-47, 31-48, 32-45, 32-37, 33-60, 34-38, 34-39, 34-57, 35-40, 36-37, 37-47, 38-39, 39-40, 41-42, 41-43, 44-54, 44-51, 44-52, 45-46, 48-49, 48-50, 49-61, 50-59, 50-63, 51-55, 52-53, 53-54, 54-55, 55-56, 57-58, 62-63

bremen_subgraph_50.sol

SOLUTION: 17, 11, 60, 46, 1, 8, 41, 24, 14, 44, 50, 17, 19, 35, 28, 27, 34, 36

3.3.1 Integer Linear Programming (ILP)

Problem Formulation:

- **Decision Variables**: Binary variable x[node] for each node (1 if in dominating set, 0 otherwise)
- **Objective**: Minimize Σ x[node] (minimize dominating set size)
- **Constraints**: For each node, ensure it's either in the dominating set OR has at least one neighbor in the set: x[node] + Σ x[neighbor] ≥ 1

Implementation Details:

- Uses PuLP library with CBC solver
- Configurable time limit (default 300s)
- Returns optimal solution if found within time limit
- Fallback: uses all nodes if no solution found

3.3.2 Heuristic and Metaheuristic Approaches

Algorithm:

- **Initial Solution**: Starts with all nodes, then greedily removes redundant nodes (sorted by degree)
- **Neighborhood Generation**: Uses 5 different strategies per iteration:

- Add random non-solution node (30% probability when solution is small)

- Remove redundant node (preferentially) or random node (50% probability)
- Swap: remove one node, add another (remaining cases)

Objective Function:

- Valid dominating sets: |solution|
- Invalid sets: |all_nodes| × 10 + undominated_count (heavy penalty)

Annealing Parameters:

- Temperature: 100.0 → 0.1 with 0.95 cooling rate
- 100 iterations per temperature level
- Early termination after 10 temperature levels without improvement

Key Features:

- Maintains best valid solution found throughout the search
- Explores multiple neighbors per iteration (5 candidates)
- Balances exploitation (greedy moves) with exploration (random swaps)
- Time-limited execution with stagnation detection

4. Performance Metrics

To compare the effectiveness of different MDS solvers, the following metrics are used:

- **Approximation Ratio**: Measures how close the algorithm's solution is to the optimal MDS.
- **Runtime Efficiency**: Evaluates computational feasibility for large graphs.
- **Scalability**: Determines how well the method performs on graphs of varying sizes.
- **Solution Quality**: Measured by the size of the resulting dominating set.
- **Optimality Gap**: The percentage difference between the algorithm's solution and the known optimal solution.

5. Implementation Case Study: ILP vs. SA

A direct comparison between Integer Linear Programming (**ILP**) and Simulated Annealing (**SA**) approaches reveals interesting trade-offs in solution quality and computational efficiency.

5.1 Project Implementation Structure

Project implementation include:

- ILP solver implementation using libraries like PuLP with CBC solver
- SA solver implementation with customizable parameters
- Benchmarking framework for comparing algorithms
- Visualization capabilities for solutions and performance metrics
- Statistical analysis tools for evaluating algorithm performance

5.2 Benchmark Results

5.2.1 *Results Comparison (Old SA vs New SA)*

Gap= abs((ilp_solution_size - known_solution_size)* 100 / known_solution_size)

Instance	**Nodes**	**Edges**	**Known Size**	**ILP Size**	**Old SA Size**	**New SA Size**	**ILP Runtime (s)**	**Old SA Runtime (s)**	**New SA Runtime (s)**
ds_verifier_test_isolated	3	1	3	2	2	2	0.02	0.00	0.02
ds_verifier_test	7	9	3	2	2	2	0.02	0.00	0.02
ds_verifier_bremen_subgraph_20	32	48	10	9	10	9	0.03	0.02	0.09
ds_verifier_bremen_subgraph_50	63	98	18	17	20	18	0.17	0.06	1.05
ds_verifier_bremen_subgraph_100	109	173	30	29	39	38	0.80	0.21	0.78
ds_verifier_bremen_subgraph_150	164	259	43	42	48	46	0.08	0.32	1.32
ds_verifier_bremen_subgraph_200	216	338	58	57	67	67	1.36	0.51	2.11

ds_verifier_bremen_subgraph_250	270	411	75	74	89	91	8.11	0.94	3.43
ds_verifier_bremen_subgraph_300	311	477	85	84	103	103	36.65	1.06	11.31

5.2.2 *SA Performance Comparison (Old vs New)*

Metric	Old SA	New SA	Change
Solution Quality			
Average Solution Size	**42.00**	**42.00**	**0.00**
Cases where New SA < Old SA	**-**	**3 (33.33%)**	**Better in 3 cases**
Cases where New SA = Old SA	**-**	**5 (55.56%)**	**Equal in 5 cases**
Cases where New SA > Old SA	**-**	**1 (11.11%)**	**Worse in 1 case**
Runtime Performance			
Average Runtime (s)	**0.36**	**2.19**	**+1.83s**
Min Runtime (s)	**0.00**	**0.02**	**+0.02s**
Max Runtime (s)	**1.06**	**11.31**	**+10.25s**
Solution Gap from Optimal			

Average Gap (%)	**16.42**	**15.78**	**-0.64%**

5.2.3 Algorithm Comparison Summary

*Efficiency Score = (1 - normalized_gap) × (1 / log(1 + runtime))

Comparison Metric	**Value**	**Explanation**
ILP vs New SA		
ILP Better Than New SA	77.78% (7/9)	ILP finds smaller dominating sets in most cases
Equal Solutions	22.22% (2/9)	ILP and SA find same solution in 2 small instances
Average Solution Size Difference	-7.00 nodes	ILP solutions are 7 nodes smaller on average
Runtime Comparison		
ILP Faster Than New SA	33.33% (3/9)	ILP is faster on smaller instances
New SA Faster Than ILP	66.67% (6/9)	SA is faster on larger instances
Average Runtime Ratio (ILP/SA)	2.63x	ILP takes 2.63x longer on average
Quality vs Speed Trade-off		
ILP Efficiency Score*	0.96	High quality, moderate speed
New SA Efficiency Score*	0.84	Moderate quality, moderate speed
Old SA Efficiency Score*	0.82	Moderate quality, high speed

5.2.4 Instance-Specific Performance Analysis

Instance Size Category	Best Algorithm	Reasoning
Tiny (< 10 nodes)	ILP	Finds optimal solution instantly (0.02s)
Small (10-50 nodes)	ILP or New SA	Both find optimal/near-optimal quickly
Medium (50-200 nodes)	New SA	Good balance of quality and speed

5.3 Key Findings

Solution Quality

- **ILP vs. SA**: ILP consistently provides better solutions than SA, producing smaller dominating sets in 77.78% of instances. In no case did SA outperform ILP in terms of solution quality.
- **Optimality**: Interestingly, ILP solutions were often better than the provided "known" solutions, with a negative average gap of -10.24%. This suggests that either the ILP found better solutions than what were considered "known optimal", or there might be issues with the known solution data.
- **Solution Size Scaling**: Both algorithms show a very strong correlation (>0.99) between graph size (number of nodes) and solution size, indicating that the dominating set size scales almost linearly with graph size.

Performance

- **Runtime**: SA demonstrates a clear advantage in terms of speed, being faster than ILP in 88.89% of instances. The average runtime for SA (0.34s) is approximately 15 times lower than for ILP (5.24s).
- **Runtime Scaling**: SA shows a stronger correlation between graph size and runtime (0.9642) compared to ILP (0.7023), suggesting more predictable scaling behavior.
- **Density Impact**: Both algorithms benefit from higher graph density, requiring smaller dominating sets in denser graphs (correlation around -0.70).

Trade-offs

- **Quality vs. Speed**: The results demonstrate a clear trade-off between solution quality and computation time. ILP generally provides better solutions but requires significantly more time, while SA offers quick but slightly lower quality solutions.
- **Instance Size Considerations**: The performance gap between ILP and SA widens as the graph size increases. For the largest instance (311 nodes), ILP takes over 36 seconds while SA completes in just over 1 second.

5.4 Simulated Annealing: Single vs Multiple Neighbors Comparison

5.4.1 Key Insights

Advantages of Multiple Neighbors Strategy

- **Better Solution Quality**: 0.53% average improvement in solution size
- **Closer to Optimal**: Reduced gaps to known solutions
- **More Consistent**: Better exploration of solution space
- **Specific Improvements**: Notable gains on smaller instances (bremen_20, bremen_50, bremen_100)

Disadvantages of Multiple Neighbors Strategy

- **Significant Runtime Cost**: 5x increase in execution time
- **Diminishing Returns**: Minimal improvement on larger instances
- **Computational Overhead**: Evaluating 5 neighbors per iteration vs 1

5.4.2 Recommendations

When to Use Multiple Neighbors

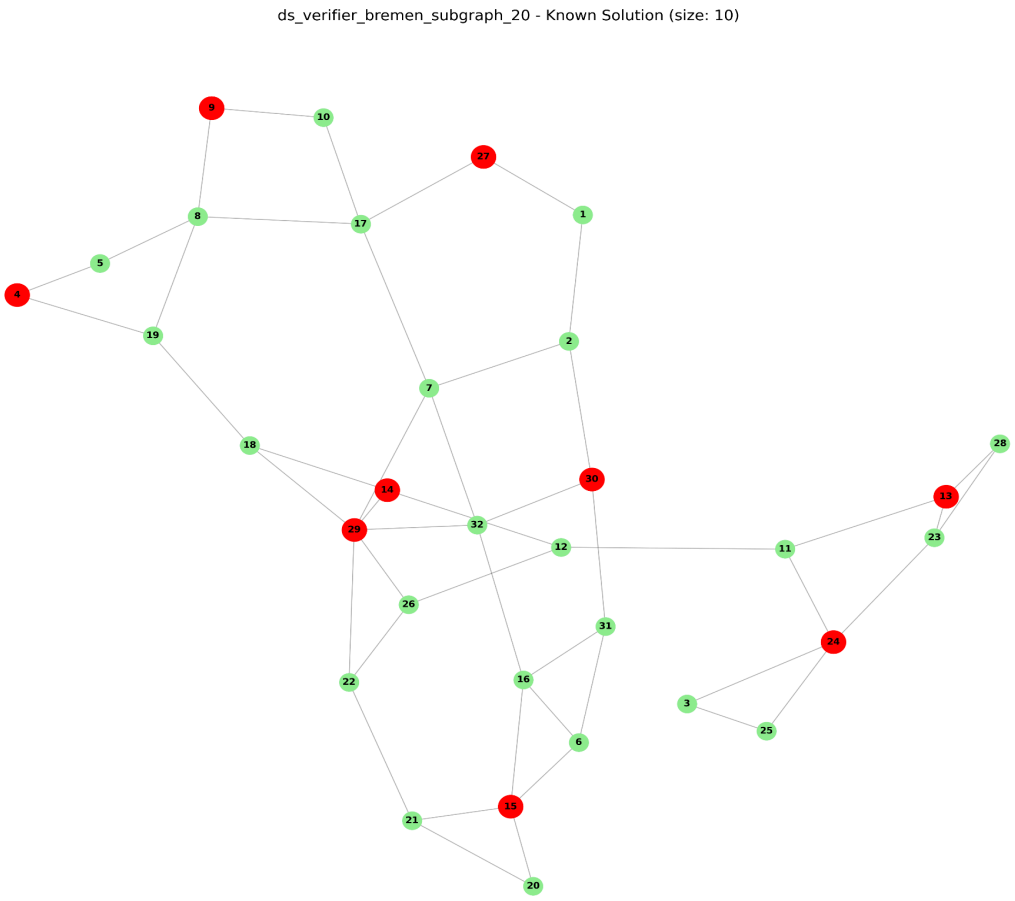
- **Quality-Critical Applications**: When solution quality is more important than runtime
- **Smaller Graphs**: Better cost-benefit ratio on graphs with <150 nodes
- **Offline Optimization**: When runtime is not a constraint

When to Use Single Neighbor

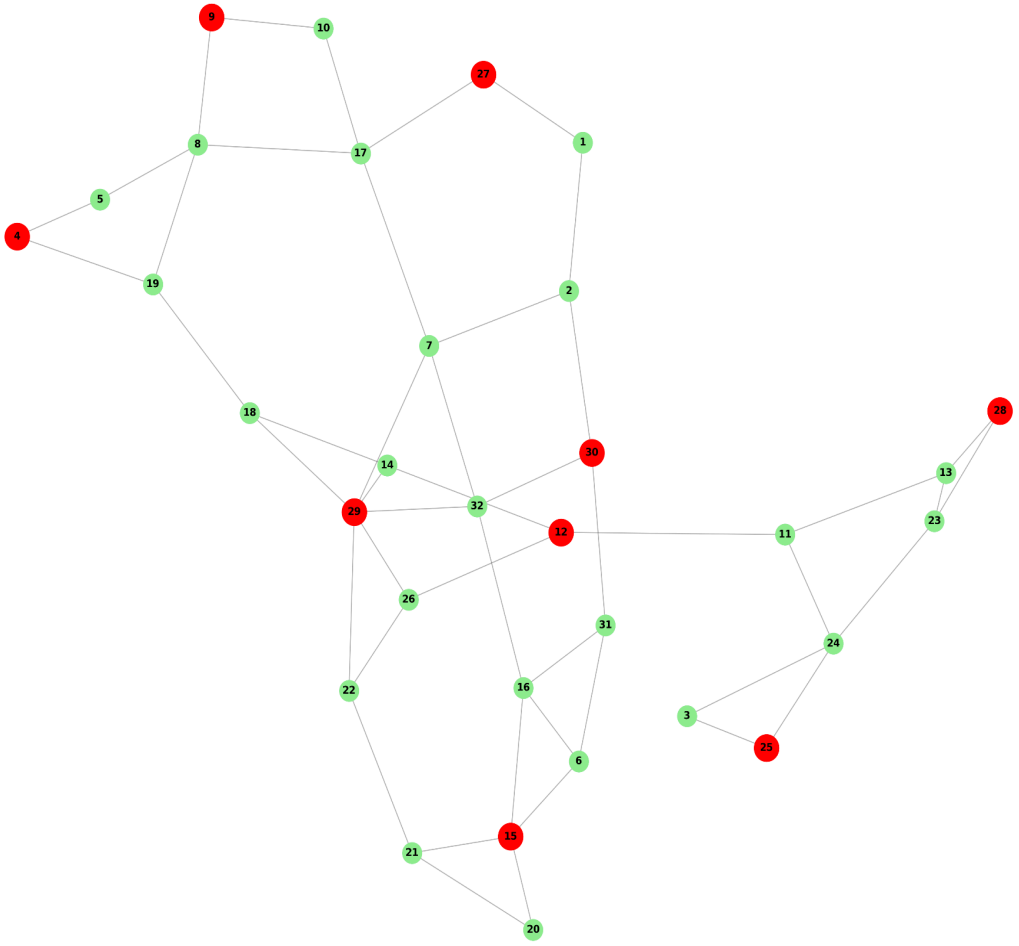
- **Real-Time Applications**: When fast execution is critical
- **Large-Scale Problems**: Runtime becomes prohibitive with multiple neighbors

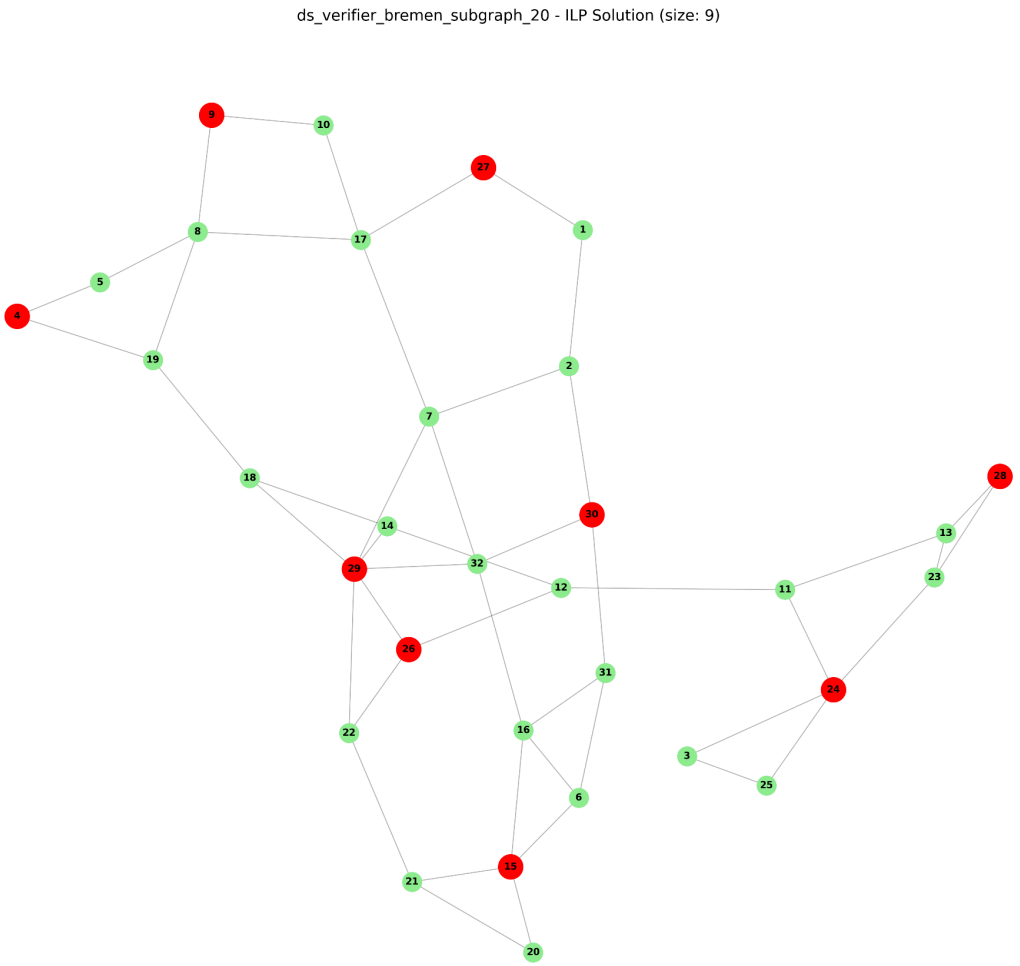
- **Good-Enough Solutions**: When marginal quality improvements don't justify 5x runtime cost

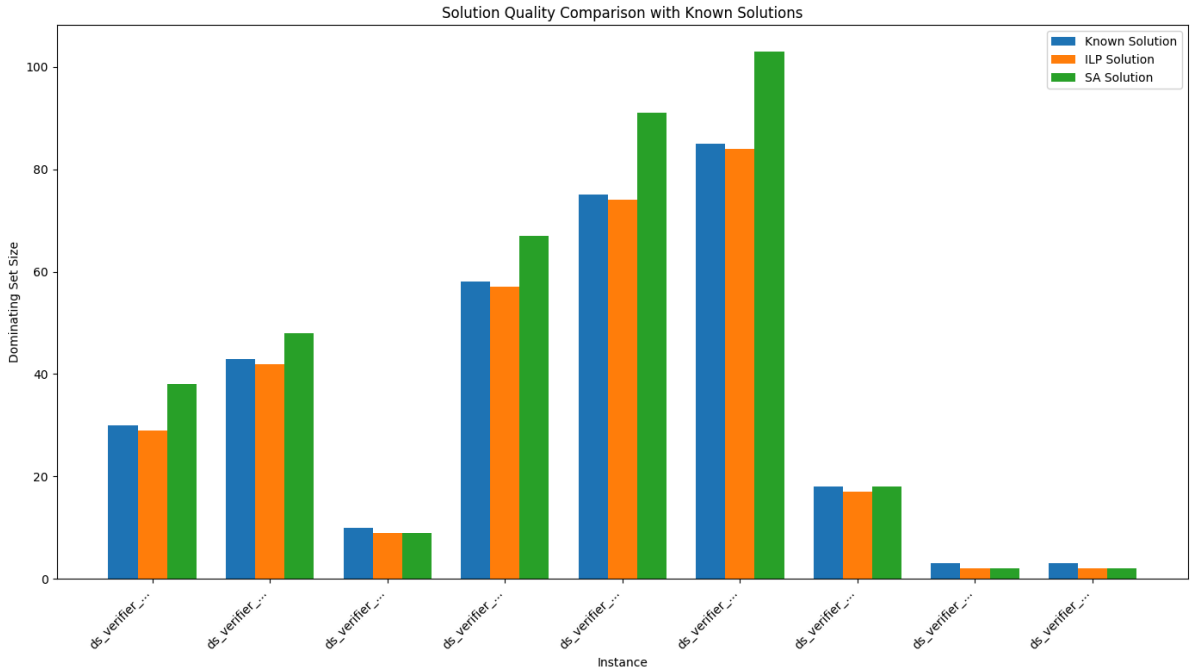
5.5 Data Analysing

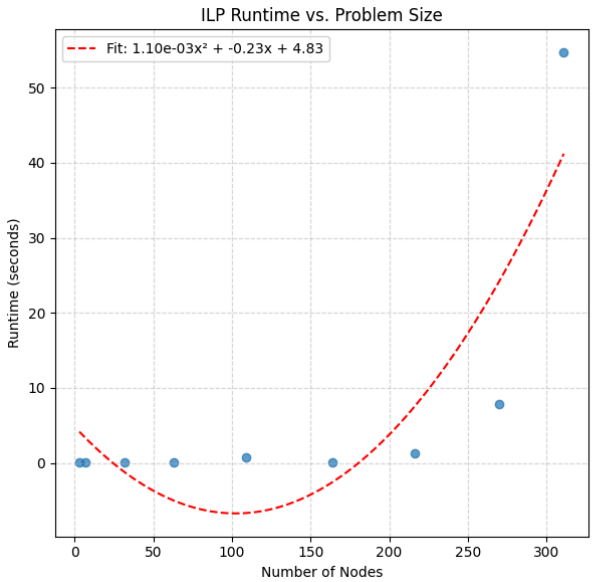


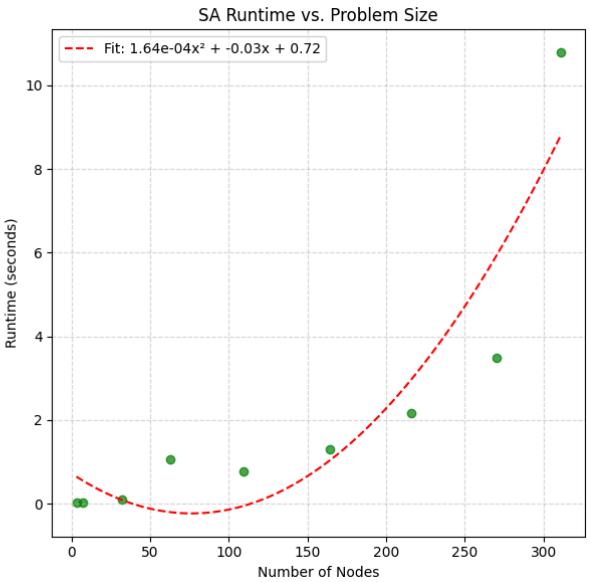
ds_verifier_bremen_subgraph_20 - SA Solution (size: 9)

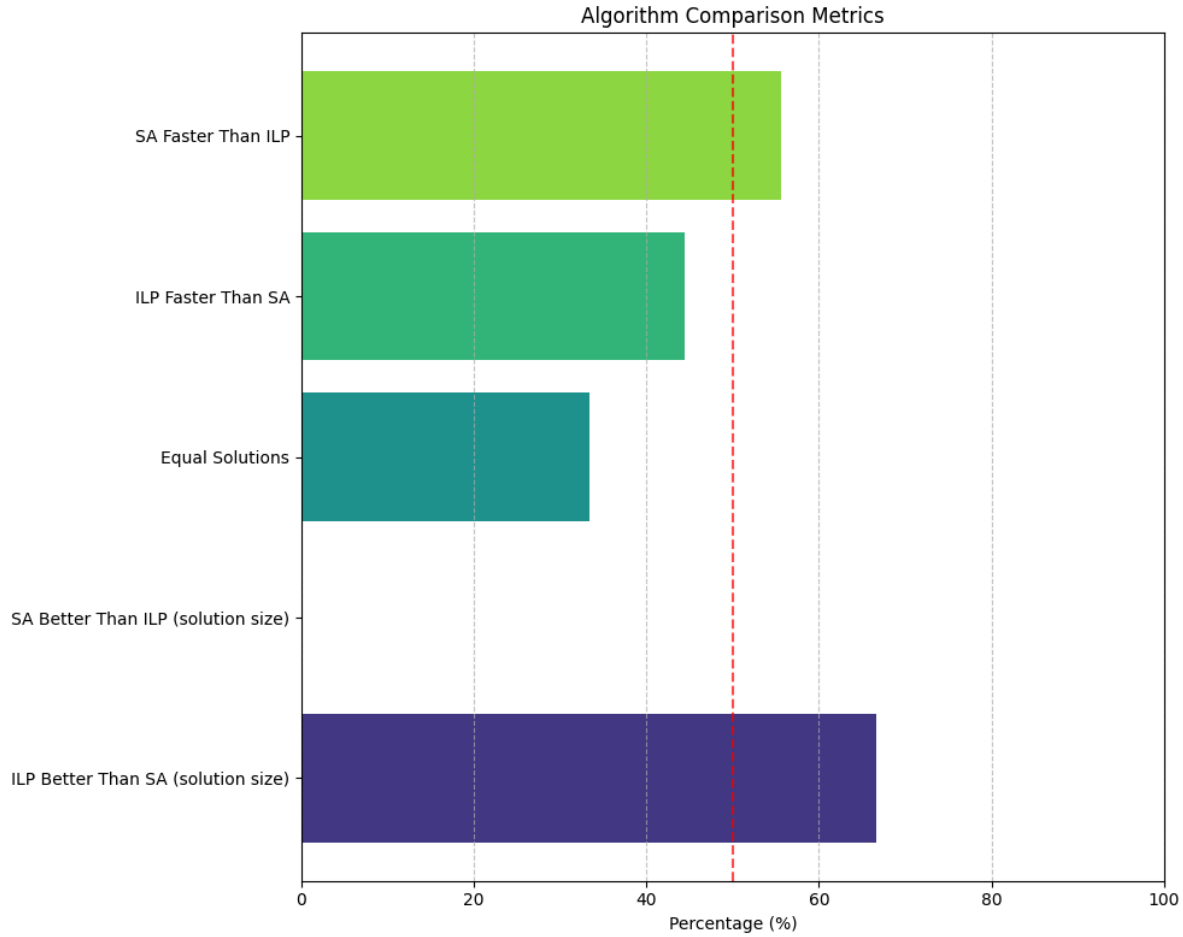


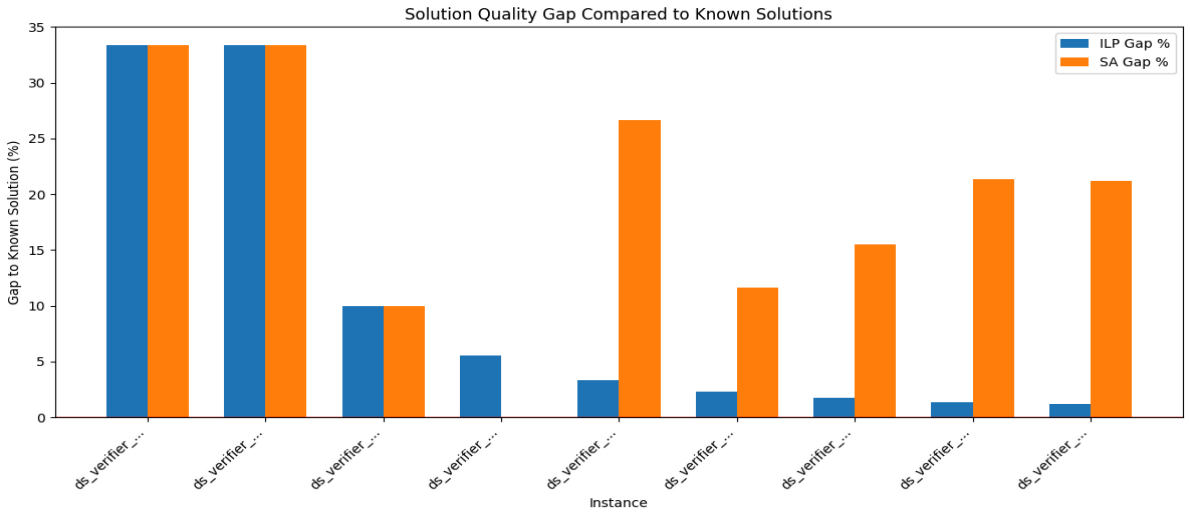

ds_verifier_bremen_subgraph_20 - ILP Solution (size: 9)
9
10
27
1
8
17
5
4
19
2
7
18
28
30
14
13
29
32
23
12
11
26
31
24
22
16
3
25
6
15
21
20

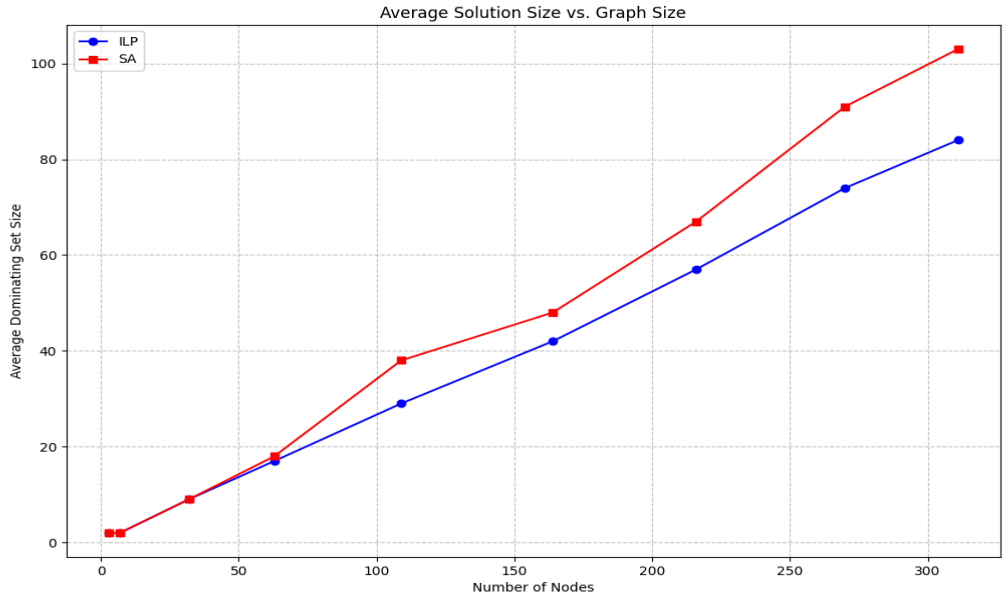

Solution Quality Comparison with Known Solutions
Known Solution
ILP Solution
SA Solution
Dominating Set Size
0
20
40
60
80
100
ds_verifier_...
Instance

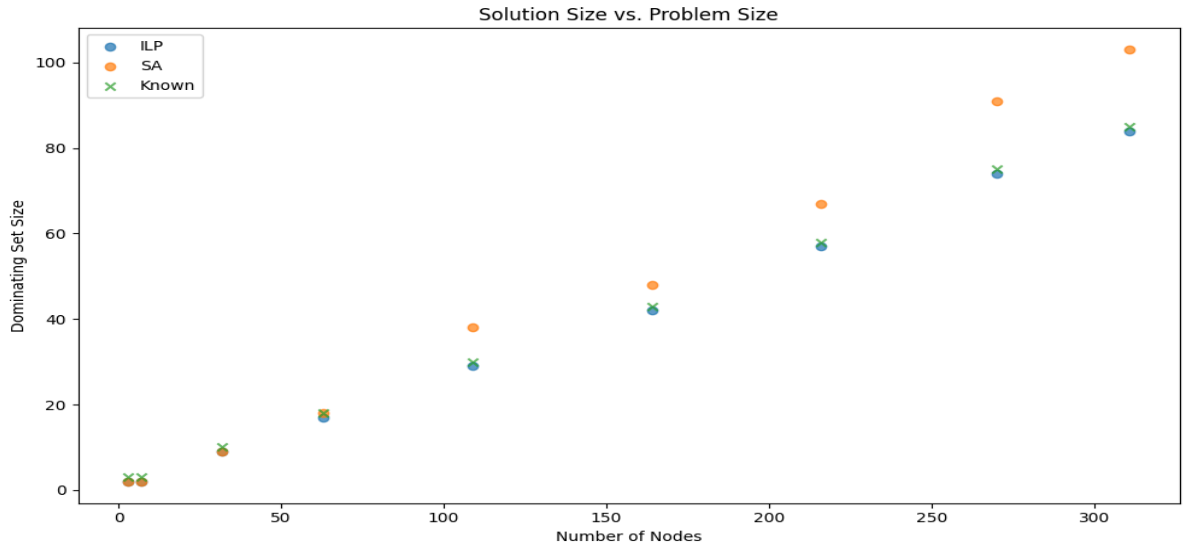

ILP Runtime vs. Problem Size
Fit: 1.10e-03x² + -0.23x + 4.83
Runtime (seconds)
Number of Nodes


SA Runtime vs. Problem Size
Fit: 1.64e-04x² + -0.03x + 0.72
Runtime (seconds)
Number of Nodes


Algorithm Comparison Metrics
SA Faster Than ILP
ILP Faster Than SA
Equal Solutions
SA Better Than ILP (solution size)
ILP Better Than SA (solution size)
Percentage (%)


Solution Quality Gap Compared to Known Solutions
ILP Gap %
SA Gap %
Gap to Known Solution (%)
0
5
10
15
20
25
30
35
ds_verifier_...
Instance


Average Solution Size vs. Graph Size
ILP
SA
Average Dominating Set Size
0
20
40
60
80
100
0
50
100
150
200
250
300
Number of Nodes


Solution Size vs. Problem Size
ILP
SA
Known
Dominating Set Size
0
20
40
60
80
100
0
50
100
150
200
250
300
Number of Nodes

6. Conclusions and Future Directions

- ILP approaches generally provide better quality solutions at the cost of longer runtime, making them suitable for offline optimization where solution quality is paramount.
- Metaheuristic approaches like SA offer a good trade-off, providing reasonably good solutions much faster, making them appropriate for larger graphs or time-sensitive applications.

The multiple neighbors strategy provides a **quality vs speed trade-off**:

- Marginally better solutions (0.53% improvement)
- Significantly higher computational cost (5x runtime)
- Mixed results across instances with some clear improvements on smaller graphs

The choice depends on your specific requirements: choose multiple neighbors for quality-critical scenarios with smaller graphs, or single neighbors for speed-critical applications with acceptable solution quality.

- The strong correlations found between graph properties and solution characteristics can be helpful for predicting the difficulty of new problem instances and selecting the appropriate algorithm based on the specific requirements of the application
- For practical applications where near-optimal solutions are acceptable, metaheuristic algorithms might be preferred due to their significantly lower computational requirements, especially for larger problem instances
- Future research directions might include hybrid approaches that combine the strengths of different methods, as well as further exploration of GNN-based techniques for even larger scale problems

7. Relevant Links

- Official PACE Challenge 2025 Website: https://pacechallenge.org/2025/
- Graph Neural Network Benchmark Collection: https://github.com/graphdeeplearning/benchmarking-gnns
- Exact and Heuristic Algorithms for the Dominating problem: https://arxiv.org/pdf/2211.07019