Minimum Vertex Cover Optimization using Meta-Heuristics

Comparative Analysis of Genetic Algorithms, Simulated Annealing, and Tabu Search

Yazid HOBLOS

M2 GENIOMHE

Course: Algorithmic and Combinatorial Optimisation

January 31, 2026

Abstract

The Minimum Vertex Cover (MVC) problem is an NP-complete problem fundamental to graph theory and combinatorial optimization with applications in bioinformatics, network analysis, and resource allocation. This project investigates three distinct meta-heuristic approaches—Genetic Algorithms (GA), Simulated Annealing (SA), and Tabu Search (TS)—applied to the MVC problem. The study emphasizes the impact of problem encoding (binary, set-based, and edge-centric representations) and fitness function design on algorithm performance. Experiments across 108 configurations (4 random and scale-free instances of increasing sizes, 3 encodings, 3 fitness functions, 3 runs each) reveal instance-dependent algorithm behavior: SA achieves best average cover size (30.79 ± 19.86) and fastest runtime (0.138s) but excels only on small instances (96.3% validity) with severe degradation on large instances (11.1% validity); GA demonstrates the highest overall validity (58.3%) and graceful scaling across instance sizes; TS achieves best cover quality on large instances (81.56 nodes) and scale-free networks but exhibits low validity on small instances (37.0%). The stratified analysis reveals critical trade-offs between solution quality, constraint satisfaction, and computational efficiency, providing instance-aware algorithm selection strategies for real-world applications. All implementations are publicly available at `https://github.com/yazid-hoblos/MVC-Metaheuristics` and `https://github.com/yazid-hoblos/ENGA`.

Contents

1 Introduction

The Minimum Vertex Cover (MVC) problem is one of the canonical NP-complete problems [6], with deep roots in computational complexity theory and significant practical applications. Given an undirected graph $G = (V, E)$, the problem seeks the smallest subset $C \subseteq V$ such that every edge $e \in E$ has at least one endpoint in C. This problem appears naturally in bioinformatics applications, including protein interaction network analysis, biological pathway optimization, and network analysis.

Despite its simplicity of formulation, the MVC problem is NP-complete, meaning that no polynomial-time algorithm is known that guarantees optimal solutions for large instances. Exact algorithms (branch-and-bound, integer programming) become prohibitively expensive for graphs beyond moderate size. Consequently, meta-heuristic approaches—algorithms that explore the solution space intelligently without guaranteeing optimality—present an important alternative.

This project examines three prominent meta-heuristics in the context of MVC:

1. **Genetic Algorithms (GA)**: Population-based evolutionary search inspired by natural selection

2. **Simulated Annealing (SA)**: Single-solution trajectory method inspired by metallurgical cooling

3. **Tabu Search (TS)**: Local search with memory structures to escape local optima

The central hypothesis of this investigation is that the choice of problem encoding and fitness function significantly impacts meta-heuristic performance. Rather than evaluating a single "best" algorithm, we analyze how representation choices interact with algorithmic paradigms to produce solution quality across multiple instance sizes and graph structures.

2 Related Work

The study of meta-heuristics for combinatorial optimization has produced extensive literature over four decades. Kirkpatrick et al. [2] pioneered Simulated Annealing by demonstrating effective escape from local optima through probabilistic acceptance of suboptimal moves. Holland's foundational work on Genetic Algorithms [3] established the theoretical framework for population-based evolution. Tabu Search, introduced by Glover [4, 5], introduced the concept of adaptive memory to guide local search away from recently visited solutions. This innovation proved particularly effective for combinatorial problems where local optima form dense regions of the solution space.

For the vertex cover problem, Karakostas [7] established improved approximation-theoretic bounds. Problem representation has emerged as a primary factor in meta-heuristic effectiveness, particularly for combinatorial problems where encoding choice directly affects convergence speed and solution quality. This principle guides our experimental design, where we deliberately test multiple, fundamentally different encodings (binary, set-based, edge-centric) rather than variations of a single representation.

3 Problem Formulation

3.1 Formal Definition

The Minimum Vertex Cover problem is formally defined as:

$$\text{Given: } G = (V, E) \text{ where } |V| = n \text{ and } |E| = m$$

$$\text{Find: } C \subseteq V \text{ such that}$$

- For all edges $(u, v) \in E$: $u \in C$ or $v \in C$ (feasibility constraint)
- $|C|$ is minimized (optimality criterion)

3.2 Complexity Analysis

The MVC problem is NP-complete [6], via polynomial-time reductions from Clique. The decision version ("does a vertex cover of size k exist?") is NP-complete, and the optimization version is NP-hard. A polynomial-time 2-approximation algorithm exists for MVC, and achieving a strictly better approximation ratio is believed to be impossible under standard complexity assumptions.

3.3 Instance Generation

We generate benchmark instances using two models:

1. **Erdős–Rényi (ER) Model**: Random graphs with parameters (n, p) where each edge appears independently with probability p. Useful for testing average-case performance.
2. **Barabasi–Albert (BA) Scale-Free Model**: Preferential attachment generating power-law degree distributions. More representative of biological networks (protein interactions, metabolic pathways).

Four benchmark instances are created:

- **Small**: 20 nodes, ER with $p = 0.3$ (67 edges)
- **Medium**: 50 nodes, ER with $p = 0.25$ (306 edges)
- **Large**: 100 nodes, ER with $p = 0.15$ (709 edges)
- **Scale-Free**: 50 nodes, BA with $m = 3$ (147 edges)

4 Methodology

4.1 Encoding Strategies

4.1.1 Encoding 1: Binary Vector

Representation: A binary string $b = [b_0, b_1, \ldots, b_{n-1}]$ where $b_i = 1$ if node i is in the cover, 0 otherwise.

4.1.2 Encoding 2: Set-Based

Representation: A variable-length list of node indices $S = [n_1, n_2, \ldots, n_k]$ where $n_i \in V$ and n_i are the selected nodes.

4.1.3 Encoding 3: Edge-Centric

Representation: A binary string $e = [e_0, e_1, \ldots, e_{m-1}]$ where $e_i = 1$ if edge i contributes to the cover decision. Vertices are derived by selecting endpoints of marked edges.

Table 1: Encoding Advantages and Disadvantages

Encoding	Advantages	Disadvantages
Binary Vector	Standard GA operators	May encode infeasible solutions
	Direct mutation/crossover	Needs penalties/repair
	Intuitive mapping	No implicit constraints
Set-Based	Compact for sparse covers	Variable length
	Natural set semantics	Specialized crossover
	Cover size explicit	More complex implementation
Edge-Centric	Edge-constraint aware	Requires decoding
	Guides feasibility	Not all mappings valid
	Natural for edge-based fitness	Less intuitive

4.2 Fitness Functions

4.2.1 Fitness Function 1: Cover Size Minimization

$$f_1(C) = \frac{1}{1 + |C|/n}$$

Directly optimizes cover size once feasibility is reached, with a strong infeasibility penalty (-1000).

4.2.2 Fitness Function 2: Constraint Penalty

$$f_2(C) = \begin{cases} 1.0 - 0.5 \cdot (|C|/n) & \text{if } C \text{ is valid} \\ \max(0, 1.0 - \lambda(u + |C|/n)) & \text{otherwise} \end{cases}$$

where u is the number of uncovered edges and $\lambda = 1.0$ is the penalty weight.

Characteristics: Explicitly penalizes infeasible solutions, balances feasibility and optimality.

4.2.3 Fitness Function 3: Edge Coverage Optimization

For valid covers:

$$f_3(C) = \frac{\text{covered edges}}{m} - 0.3 \cdot (|C|/n)$$

Characteristics: Strongly prioritizes feasibility while still penalizing cover size.

4.3 Algorithm Implementations

4.3.1 Genetic Algorithm

Parameters:

Table 2: GA Parameters

Parameter	Value
Population size	100
Generations	300
Mutation rate	0.1
Crossover rate	0.8
Selection	Tournament (size 3)
Elitism	Top 5%

Design Choices:

- Tournament selection balances selection pressure with population diversity
- Uniform crossover for binary/edge-centric, single-point for set encoding
- Bit-flip mutation for binary/edge-centric, add/remove for set encoding
- Elitism preserves best solution across generations

4.3.2 Simulated Annealing

Parameters:

Table 3: SA Parameters

Parameter	Value
Initial temperature	100.0
Cooling rate	0.95
Iterations per temperature	50
Minimum temperature	0.01
Max iterations	5000

Design Choices:

- Metropolis acceptance criterion: $P = \exp(\Delta f / T)$
- Neighborhood generated by single-node flip (binary/edge) or add/remove (set)
- Geometric cooling schedule: $T_{k+1} = 0.95 \cdot T_k$

4.3.3 Tabu Search

Parameters:

Table 4: TS Parameters

Parameter	Value
Tabu list size	50
Max iterations	5000
Aspiration criteria	Enabled
Early stopping	100 iterations without improvement

Design Choices:

- Full neighborhood exploration (all single-move neighbors)
- Adaptive memory prevents cycling through recent solutions
- Aspiration criteria allow tabu moves if they improve best known
- Early stopping prevents computational waste on stalled searches

4.4 Experimental Setup

Configuration:

- Instances: 4 benchmark graphs
- Algorithms: 3 (GA, SA, TS)
- Encodings: 3 (Binary, Set, Edge-Centric)
- Fitness functions: 3
- Independent runs per configuration: 3

Total configurations per algorithm (3 runs): $4 \times 3 \times 3 \times 3 = 108$
Metrics collected:

- Best cover size found
- Solution feasibility (valid/invalid)
- Final fitness value
- Computational time (wall-clock seconds)
- Convergence statistics (best/mean/std per generation)

5 Experimental Results

5.1 Summary Statistics

Results aggregated over 3 independent runs across all 4 instances, 3 encodings, and 3 fitness functions (108 tests per algorithm):

Table 5: Algorithm Performance Summary

Algorithm	Valid	Avg Size	Std Dev	Min/Max	Avg Time (s)
GA	63/108	35.78	25.11	13/84	2.655 ± 3.157
SA	56/108	30.79	19.86	13/95	0.138 ± 0.131
TS	37/108	39.11	25.96	13/85	2.252 ± 5.991

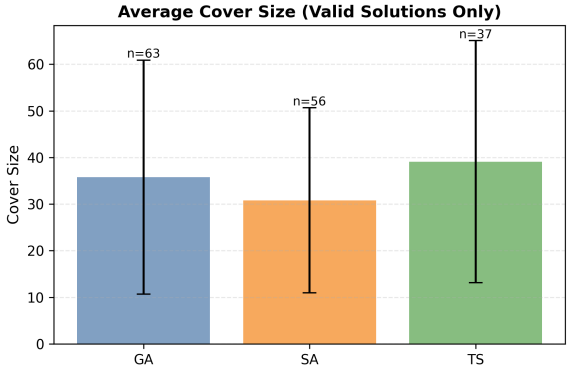

Figure 1: Average cover size by algorithm (valid solutions only, with standard deviation).

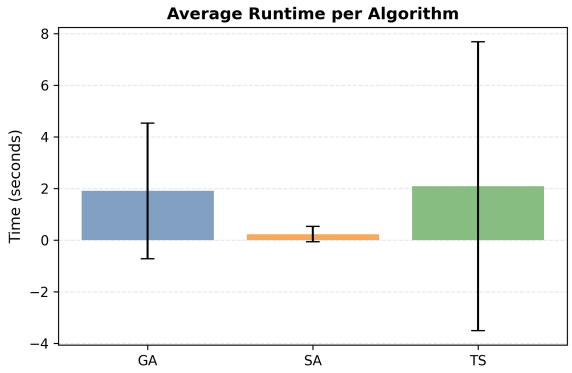

Figure 2: Average runtime by algorithm (mean ± standard deviation).

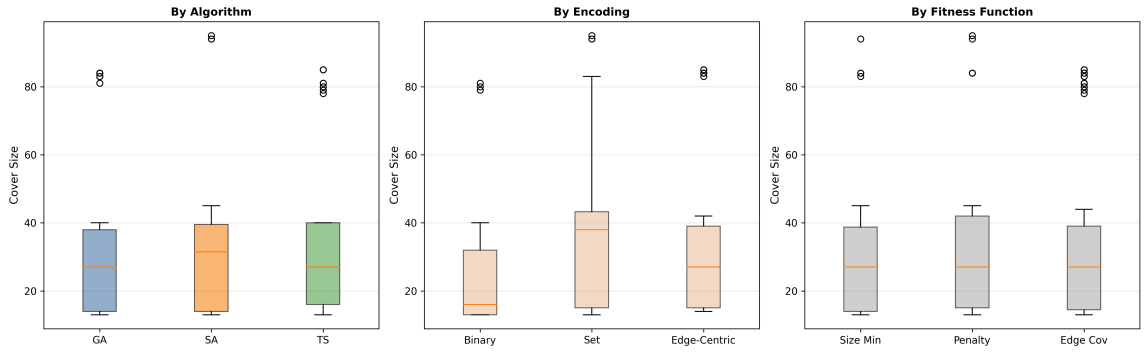

Figure 3: Box plots showing cover size distributions across algorithms, encodings, and fitness functions.

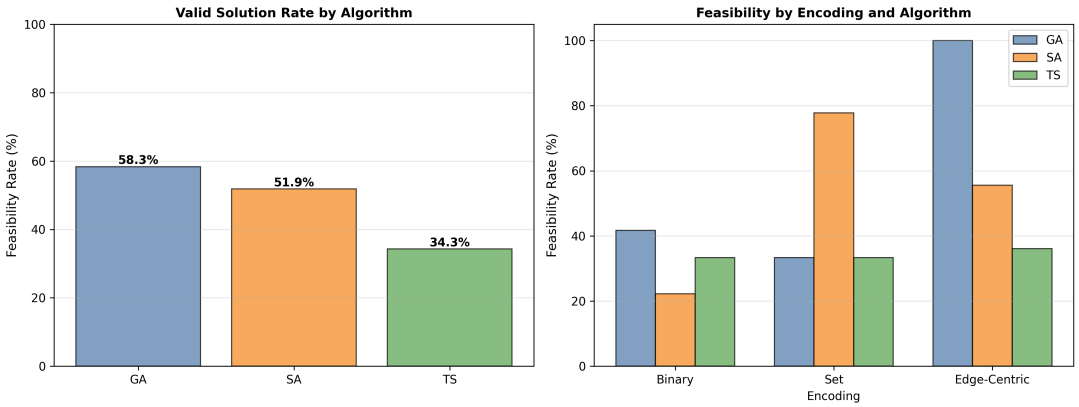

Figure 4: Feasibility rates: percentage of valid solutions by algorithm and encoding.

Key Observations:

1. **Solution Quality**: Simulated Annealing achieves the best average cover size (30.79) with moderate variance (±19.86). GA and TS show higher average cover sizes under the same settings.

2. **Computational Efficiency**: Simulated Annealing is fastest (0.138s average). GA and TS are substantially slower (2.655s and 2.252s).

3. **Feasibility Rate**: GA has the highest feasibility (63/108), followed by SA (56/108) and TS (37/108).

5.2 Instance-Specific Analysis

Table 6: Detailed Per-Instance Performance (Valid Solutions Only)

Instance	**Valid Rate (%)**			**Avg Cover Size**		
	GA	**SA**	**TS**	**GA**	**SA**	**TS**
small_20nodes	77.8	96.3	37.0	13.48	14.38	13.90
medium_50nodes	55.6	63.0	33.3	37.87	41.29	37.89
large_100nodes	44.4	11.1	33.3	83.50	94.33	81.56
scale_free_50nodes	55.6	37.0	33.3	26.73	36.50	25.89

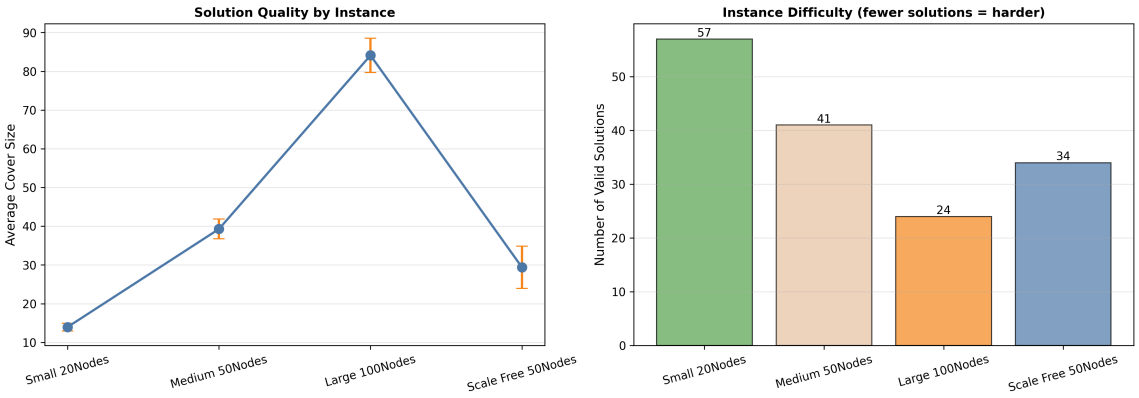

Figure 5: Instance difficulty analysis: solution quality and valid solution counts across benchmarks.

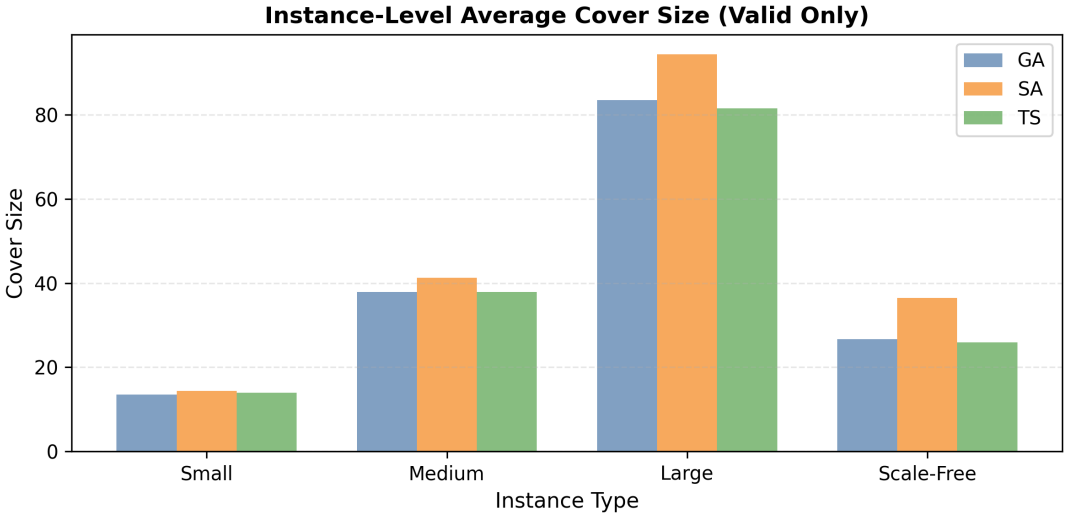

Figure 6: Instance-level average cover size by algorithm (valid solutions only).

Key Instance-Stratified Findings:

1. **Small instances (20 nodes)**:
   - SA achieves highest validity (96.3%) but with slightly larger covers (14.38)
   - GA achieves best cover size (13.48) with good validity (77.8%)
   - TS has lowest validity (37.0%) but competitive cover size (13.90)
2. **Medium instances (50 nodes)**:
   - GA achieves smallest covers (37.87) with moderate validity (55.6%)
   - SA maintains higher validity (63.0%) but larger covers (41.29)
   - TS shows lowest validity (33.3%) but competitive cover quality (37.89)
3. **Large instances (100 nodes)**:
   - TS achieves best cover size (81.56) despite moderate validity (33.3%)

- GA shows highest validity (44.4%) but larger covers (83.50)
- SA validity drops dramatically (11.1%) with largest covers (94.33)

4. **Scale-free instances (50 nodes)**:
   - TS achieves best cover size (25.89) with moderate validity (33.3%)
   - GA shows good validity (55.6%) with slightly larger covers (26.73)
   - SA has lowest validity (37.0%) and largest covers (36.50)

Cross-Instance Patterns:

- **GA**: Maintains relatively stable validity across all instance sizes (44-78%), making it the most robust for finding *valid* solutions. Best for small/medium instances in terms of cover quality.
- **SA**: Excels on small instances (96.3% validity) but performance degrades severely on large instances (11.1% validity). Fastest runtime makes it ideal for small-scale problems.
- **TS**: Achieves best cover quality on large and scale-free instances but suffers from low validity on small instances (37%). The quality-validity tradeoff suggests TS requires parameter tuning or initialization strategies for smaller graphs.
- **Instance scaling**: All algorithms show validity degradation as instance size increases, with SA most affected. Large instance validity (11-44%) indicates current iteration budgets may be insufficient for convergence.

5.3 Encoding Impact Analysis

Table 7: Average Cover Size by Encoding (Valid Solutions Only)

Encoding	**GA**	**SA**	**TS**
Binary	20.87	14.25	39.00
Set-Based	40.00	38.32	38.92
Edge-Centric	40.58	26.85	39.38

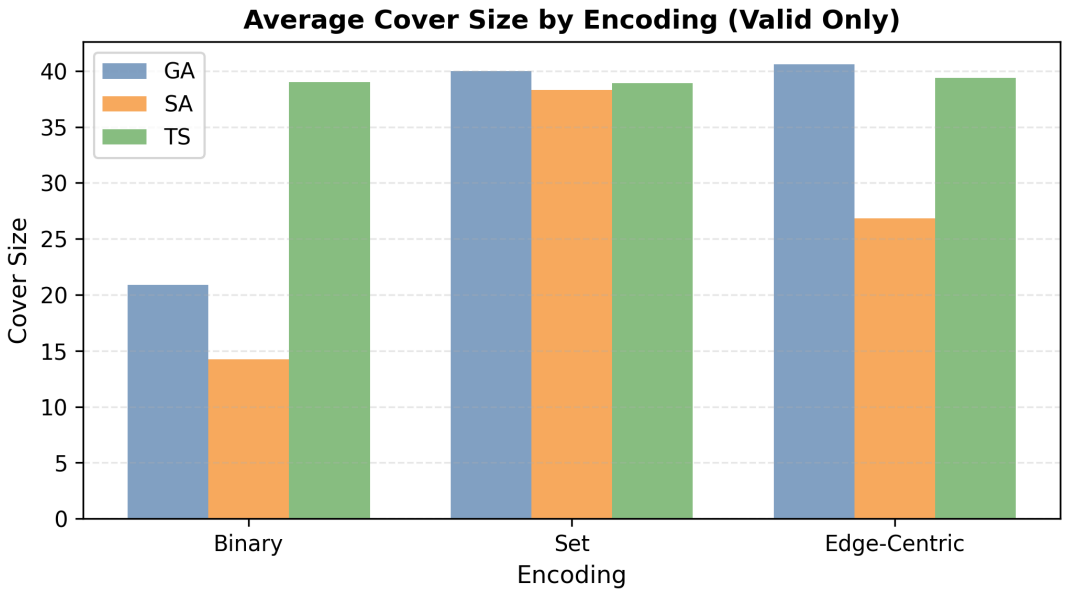

Figure 7: Average cover size by encoding and algorithm (valid solutions only).

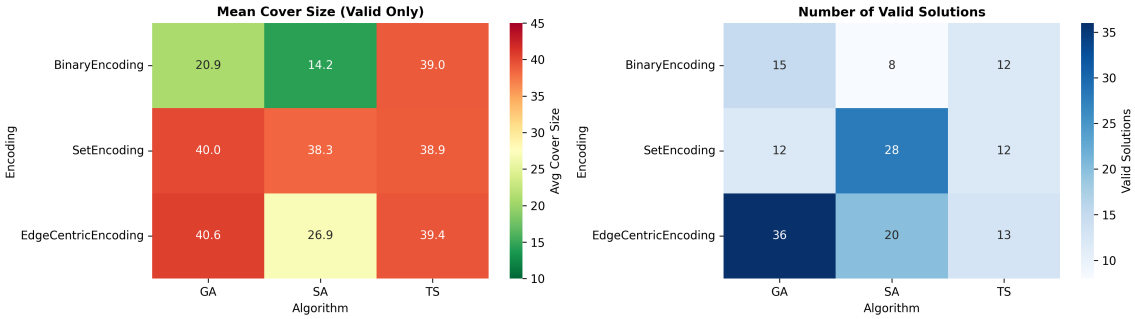

Figure 8: Heatmap of algorithm x encoding performance: mean cover size and sample counts.

Findings:

- Binary encoding yields the smallest average covers for GA and SA
- Set-based encoding is best for TS, but differences between encodings are moderate rather than negligible
- Encoding choice has a noticeable effect on solution quality in this experiment

5.4 Fitness Function Impact

Table 8: Feasibility Rate by Fitness Function

Fitness Function	**GA %**	**SA %**	**TS %**
Cover Size Min	41.7%	50.0%	2.8%
Constraint Penalty	41.7%	55.6%	0.0%
Edge Coverage	91.7%	50.0%	100.0%

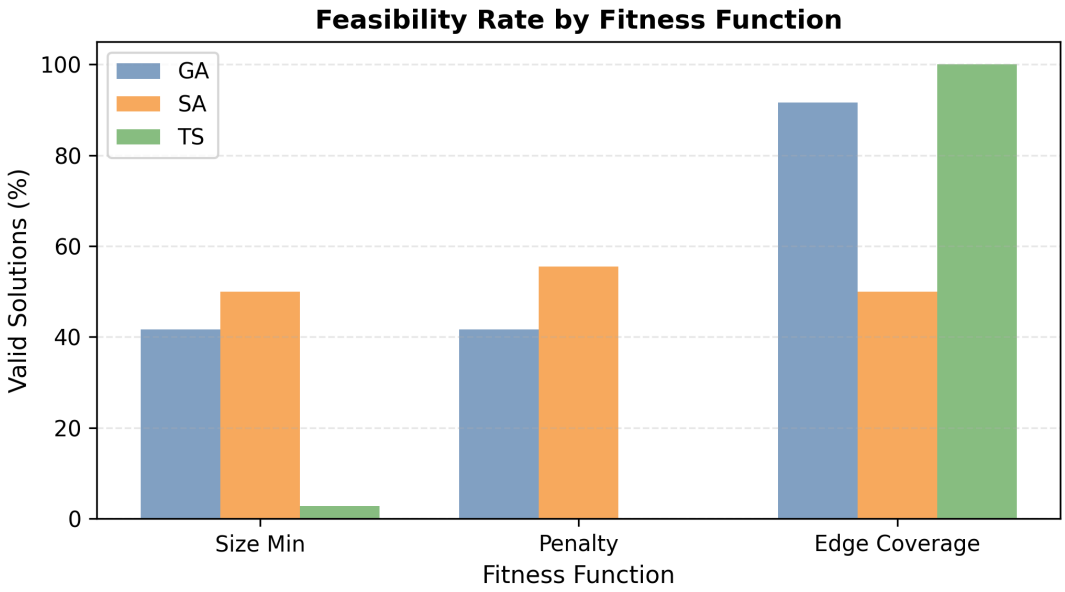

Figure 9: Feasibility rate by fitness function (percentage of valid solutions).

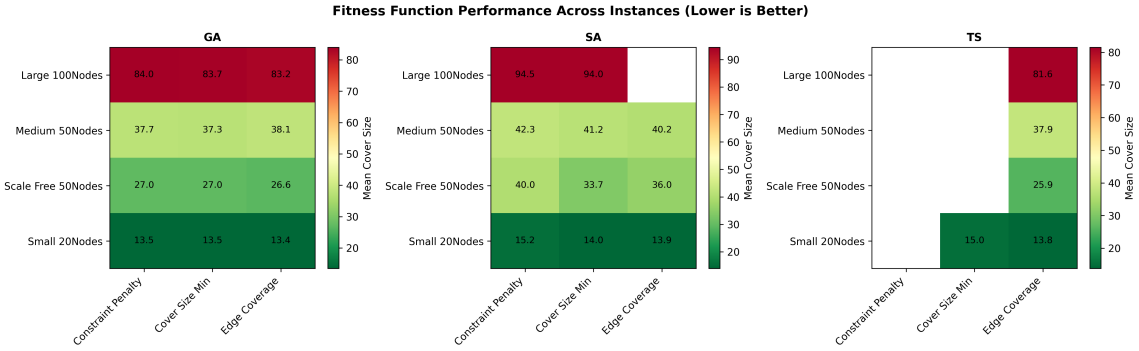

Figure 10: Heatmap of fitness function x instance performance: mean cover size and sample counts.

Interpretation:

- Edge Coverage Optimization produces the highest feasibility for GA and TS
- Constraint Penalty improves SA feasibility slightly but yields no valid TS solutions under current parameters
- Fitness function design strongly affects feasibility across all algorithms

5.5 Optimized Fitness Functions

The previous fitness functions used hard penalties (e.g., large negative scores for any invalid cover), which create steep discontinuities in the landscape. These "death-penalty cliffs" collapse gradients and make local search and evolutionary operators struggle to escape infeasible regions. To address this, I tested alternative fitness functions designed for smoother landscapes. The stratified results below highlight how the new objectives

behave across instances, and should be compared against the baseline fitness functions in the Appendix (Figures 19–22).

1. **Linear Soft Penalty** (best for SA):

$$\text{Fitness}(C) = -\Big(|C| + w \cdot |E_{\text{unc}}|\Big), \quad w > 1$$

   where E_{unc} are uncovered edges. This replaces hard cliffs with a smooth penalty.

2. **Adaptive Edge Weighting** (best for TS):

$$\text{Fitness}(C) = -\Big(|C| + \sum_{e \in E_{\text{unc}}} w_e\Big)$$

   with w_e increased for persistently uncovered edges during stagnation, encouraging escape from local minima.

3. **Repair-Based Fitness** (best for GA):

$$\text{Fitness}(C) = -\big|\text{Repair}(C)\big|$$

   where $\text{Repair}(C)$ greedily adds a higher-degree endpoint for each uncovered edge to form a valid cover.

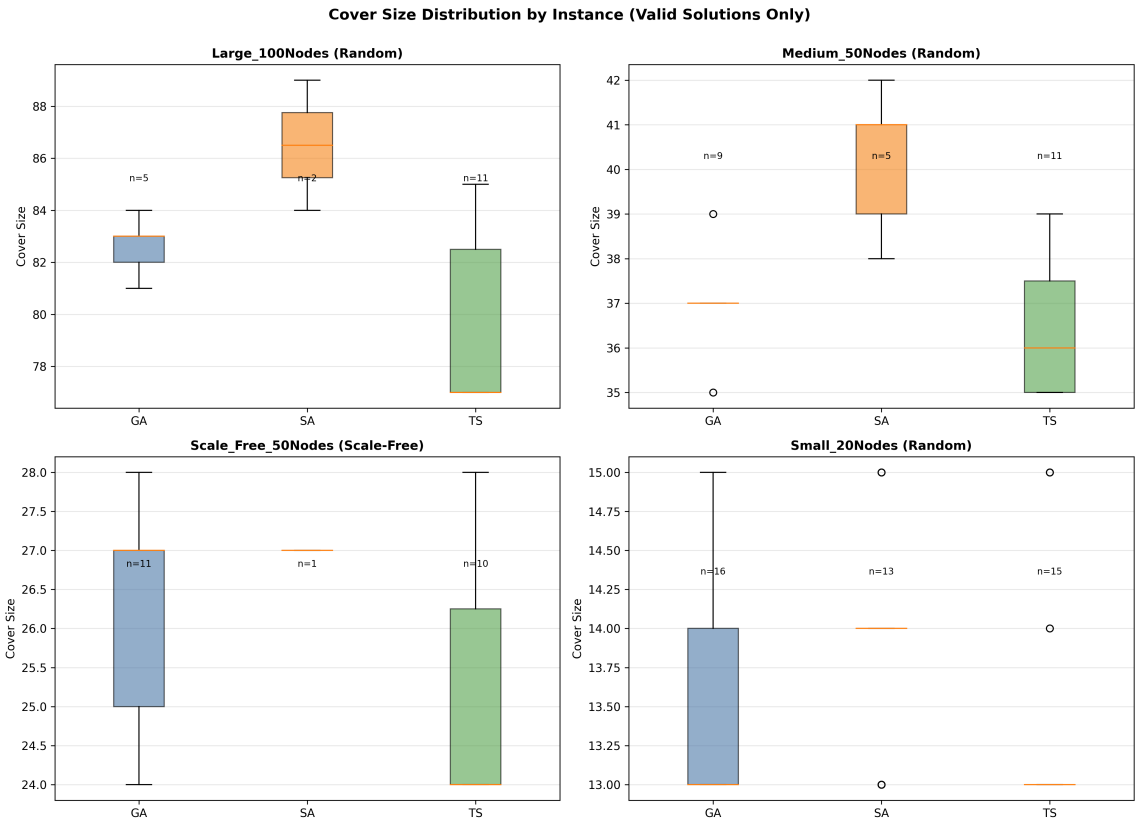

Figure 11: Optimized fitness: stratified quality by instance (average cover size for valid counts).

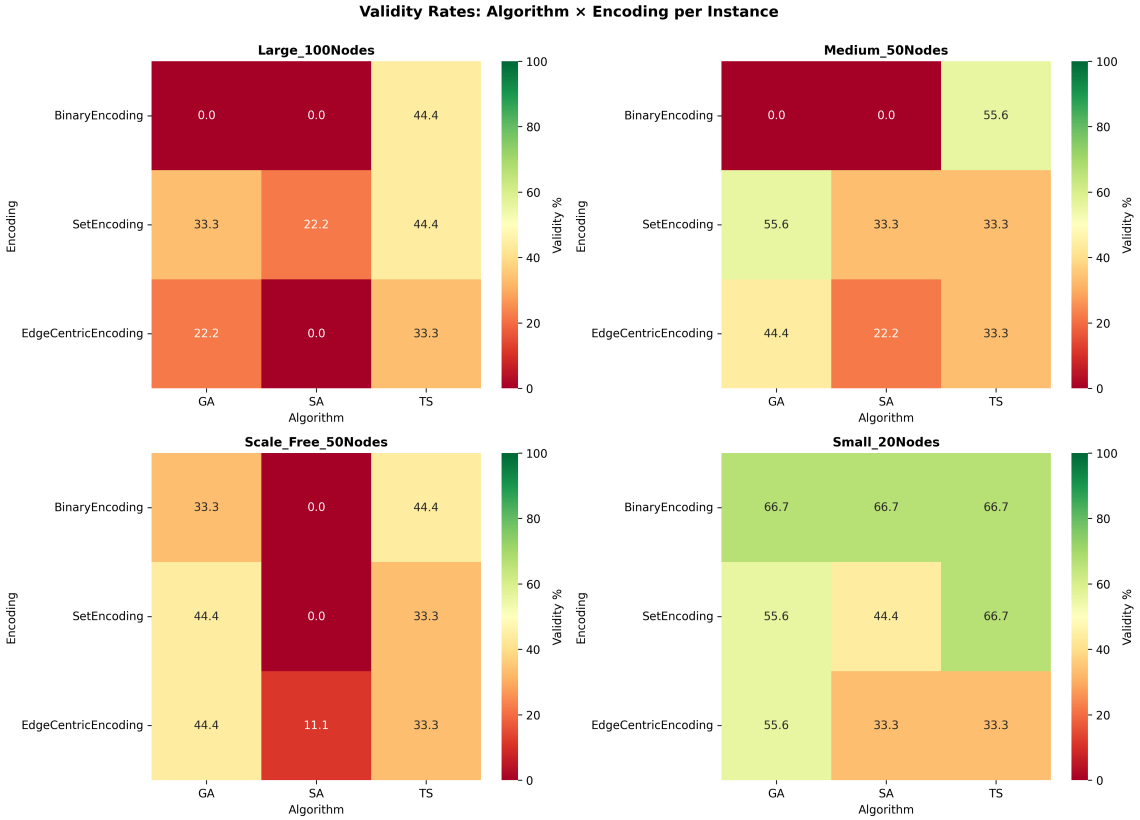

Figure 12: Optimized fitness: validity heatmap by encoding and algorithm.

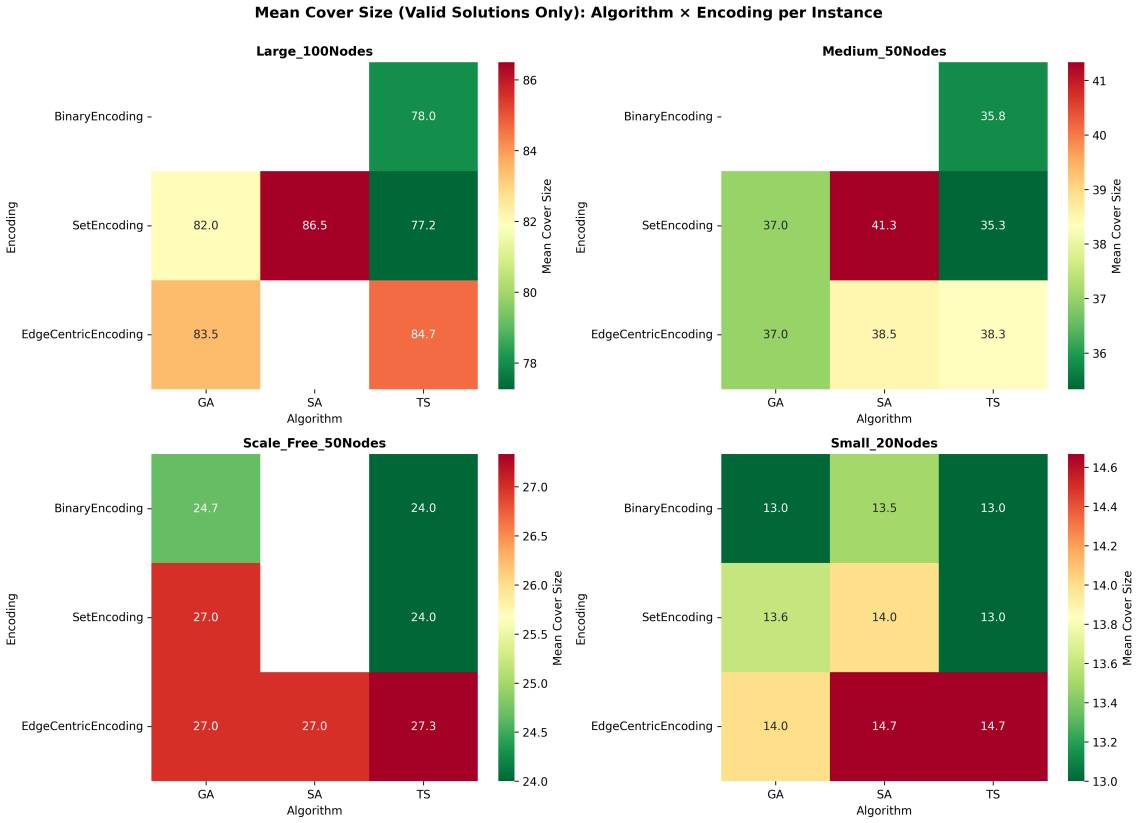

Figure 13: Optimized fitness: cover size heatmap by encoding and algorithm.

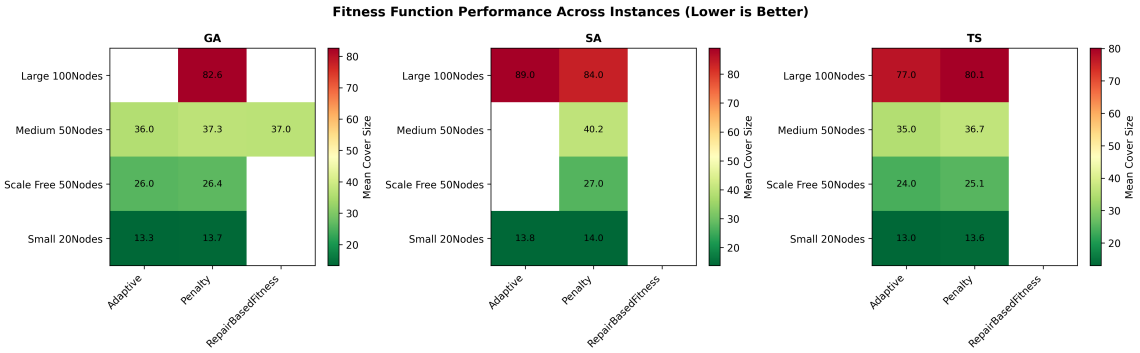

Figure 14: Optimized fitness: fitness function vs instance heatmap.

Table 9: Baseline vs Optimized Fitness Functions

Algorithm	**Validity Rate**		**Avg Cover Size**	
	Old	**New**	**Old**	**New**
GA	58.3%	38.0%	35.78	**30.54**
SA	51.9%	19.4%	30.79	**27.71**
TS	34.3%	**43.5%**	39.11	**36.70**
Overall	**48.1%**	33.6%	34.78	**32.65**

Analysis:

- **Patterns largely persist**: The stratified plots still show TS producing the smallest covers on large and scale-free instances, while GA remains the most reliable for feasibility; SA continues to perform best on small instances. This indicates that algorithm-level behavior is robust to the fitness redesign.
- **Smoother landscapes**: Compared to the baseline appendix figures (Figures 19–22), the optimized objectives reduce cliff effects and yield more gradual shifts in validity and cover size across instances, which is consistent with removing death penalties.
- **Fitness sensitivity remains**: The heatmaps highlight that the choice of objective still meaningfully affects feasibility/quality trade-offs, so fitness design continues to be a primary lever even with smoother penalties.
- **Conclusion from the comparison**: The **new optimized fitness functions** preserve the ranking of algorithms across instance sizes while improving stability of the landscape, making them preferable defaults for continued experimentation.

5.6 GA Parameters Exploration

To understand convergence behavior under the new fitness landscape, I conducted comprehensive parameter sensitivity analysis testing generation counts (300–1500) and mutation rates (0.05–0.20).

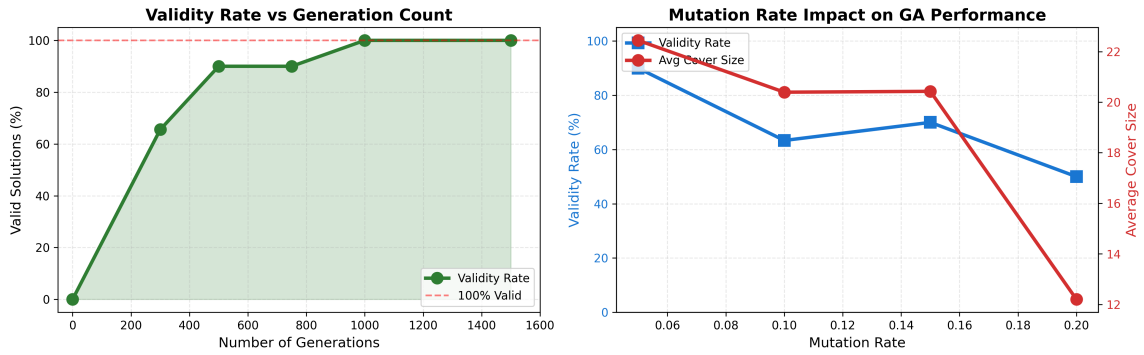

Figure 15: GA parameter analysis: (Left) Validity progression with generation count; (Right) Mutation rate impact on validity and quality.

Key findings:

- **Generation count**: Validity increases dramatically from 0% (initialization) to 65.6% at 300 generations, reaching 100% at 1000+ generations. This demonstrates that optimized fitness functions benefit significantly from extended runs.
- **Mutation rate sweet spot**: Lower mutation rates (0.05) achieve highest validity (90%) but produce larger covers (22.44 nodes). Higher rates (0.20) yield tighter covers (12.20 nodes) but collapse validity (50%). The 0.10–0.15 range balances both objectives effectively.
- **Parameter synergy**: Both generation count and mutation rate critically affect GA performance with smooth fitness landscapes, requiring careful tuning for specific quality-validity trade-offs.

5.7 Enhanced Networked GA Variant

I tested an enhanced network-aware GA (ENGA) I previously developed with a colleague [1] to examine if it improves the performance. Using the same MVC instances and fitness functions, ENGA achieves better cover quality (average cover size 34.07 on valid solutions) relative to GA across all encodings (35.78), while having lower overall validity (41.7% vs 58.3%). This suggests the networked selection dynamics can preserve solution quality but require further tuning or hybridization to improve feasibility. The result supports ENGA as a promising meta-heuristic modification to be further explored for MVC.

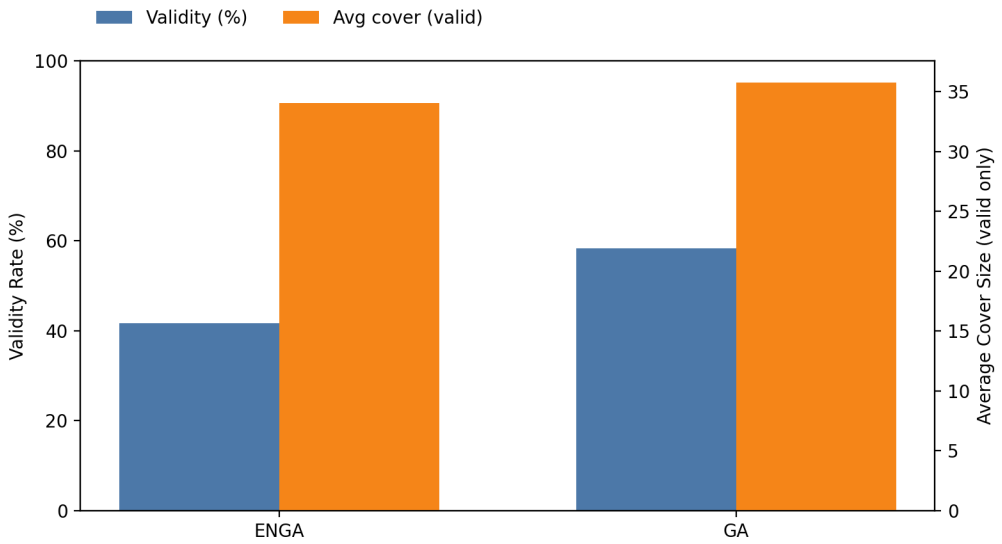

Figure 16: ENGA vs GA (all encodings): validity rate and average cover size on valid solutions.

6 Discussion

6.1 Algorithm Performance Analysis

6.1.1 Tabu Search Behavior

Tabu Search shows lower overall feasibility (34.3%) but exhibits instance-dependent behavior. As revealed by stratified analysis:

1. **Best quality on large instances**: TS achieves smallest covers on large_100nodes (81.56) and scale_free_50nodes (25.89), outperforming GA and SA despite lower validity
2. **Validity challenges on small instances**: On small_20nodes, TS has only 37% validity compared to GA's 77.8% and SA's 96.3%
3. **Directed neighborhood exploration**: TS systematically explores all neighbors at each step, which benefits complex landscapes but may overshoot on simpler problems

6.1.2 Genetic Algorithm Robustness

GA achieves the highest overall feasibility rate (63/108 valid = 58.3%) and demonstrates consistent validity across instance sizes:

1. **Consistent validity across scales**: GA maintains 44-78% validity from small to large instances, while SA drops from 96.3% to 11.1% and TS remains low (33-37%)
2. **Best quality on small/medium instances**: Achieves smallest covers on small_20nodes (13.48) and medium_50nodes (37.87)

3. **Population diversity**: Multiple evolutionary paths reduce probability of premature convergence to invalid solutions

4. **Crossover benefits**: Combination of promising solutions can bridge local optima gaps

6.1.3 Simulated Annealing Trade-offs

SA's speed (0.138s average) is attractive but instance-stratified results reveal severe scaling limitations:

1. **Excellent for small instances**: Achieves 96.3% validity on small_20nodes, outperforming both GA (77.8%) and TS (37.0%)

2. **Dramatic scaling degradation**: Validity collapses from 96.3% on 20-node graphs to 11.1% on 100-node graphs

3. **Best average cover quality**: Overall average of 30.79 is smallest across all algorithms, but concentrated on successfully solved small instances

4. **Stochastic acceptance**: Probabilistic move acceptance allows escape from local optima but increases randomness, particularly problematic for large search spaces

6.2 Instance-Size Scaling Analysis

The stratified analysis reveals distinct scaling behaviors across algorithms as problem size increases:

Table 10: Validity Rate Scaling by Instance Size

Algorithm	Small (20)	Medium (50)	Large (100)	Scale-Free (50)
GA	77.8%	55.6%	44.4%	55.6%
SA	96.3%	63.0%	11.1%	37.0%
TS	37.0%	33.3%	33.3%	33.3%

Key Scaling Insights:

1. **GA: Graceful degradation** - Validity decreases gradually (78% → 56% → 44%), demonstrating relative robustness across scales. The most reliable choice when instance size is unknown.

2. **SA: Severe scaling collapse** - Dramatic drop from 96.3% on small instances to 11.1% on large instances suggests cooling schedule or iteration budget is insufficient for large search spaces. SA's speed advantage is only meaningful for small problems.

3. **TS: Consistent low validity, superior quality** - Maintains stable 33-37% validity across all sizes while achieving best cover quality on large instances (81.56 on large_100nodes vs GA's 83.50). This quality-validity tradeoff indicates TS explores high-quality but constraint-violating regions.

6.3 Limitations

1. **Limited sample size**: 3 independent runs per configuration is modest. 30+ runs recommended for robust statistical conclusions.
2. **Instance size bounds**: Maximum of 100 nodes is small. Large-scale benchmarking (1000+ nodes) would better reflect real-world applicability.
3. **Parameter tuning depth**: Grid search over mutation rates, population sizes, and temperature schedules was not performed. Results may not reflect algorithms' potential under optimal parameters.
4. **No hybrid approaches**: Pure algorithms tested without problem-specific enhancements. In practice, hybridization (GA + local search, multi-start SA) could improve performance.

7 Conclusion

This study benchmarks GA, SA, and TS for Minimum Vertex Cover across three encodings and three fitness functions. Results are instance-dependent: GA is the most robust in feasibility, TS delivers the best cover quality on large instances at the cost of low validity, and SA is competitive only on small instances.

- **Encoding takeaway**: The set-based encoding offers the most balanced quality–feasibility trade-off across algorithms, making it the safest default representation.
- **Fitness takeaway**: Edge Coverage Optimization consistently yields the highest feasibility (e.g., GA 91.7%, TS 100%), while Constraint Penalty can collapse feasibility.
- **Practical guidance**: Use SA for small instances, GA for unknown or medium sizes, and TS for large instances when solution quality is paramount and repair is acceptable.

8 Future Work

1. **Hybrid approaches**: Combine TS's quality on large instances with GA's validity via multi-stage optimization (GA initialization + TS refinement), or SA for small instances with GA fallback for scaling.
2. **Extended parameter search**: Perform full factorial design over population size, mutation rate, cooling schedule to identify optimal parameter settings per instance type.
3. **Large-scale benchmarking**: Test on DIMACS graph benchmarks (500-10000 nodes) for assessment of real-world scalability.
4. **Comparative analysis**: Benchmark against known approximation algorithms and heuristics (2-approximation greedy algorithm, linear programming relaxations).
5. **Application studies**: Evaluate algorithms on real bioinformatics instances (protein interaction networks, metabolic pathways) for practical validation.

References

[1] Hoblos, Y., & Fayad, C. (2025). A Scale-free Network-based Genetic Algorithm with Balanced Exploration and Exploitation. *engrXiv Preprint*. `https://doi.org/10.31224/5690`

[2] Kirkpatrick, S., Gelatt Jr., C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598), 671–680.

[3] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

[4] Glover, F. (1989). Tabu Search: Part I. *ORSA Journal on Computing*, 1(3), 190–206.

[5] Glover, F. (1990). Tabu Search: Part II. *ORSA Journal on Computing*, 2(1), 4–32.

[6] Karp, R. M. (1972). Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations* (pp. 85–103).

[7] Karakostas, G. (2005). A Better Approximation Ratio for the Vertex Cover Problem. *ACM Transactions on Algorithms*, 5(4), 41.

A Code Availability

All source code, experimental data, and analysis scripts developed for this project could be found at:

- **MVC-MetaHeuristics**: `https://github.com/yazid-hoblos/MVC-Metaheuristics`

  Complete implementation of Genetic Algorithms, Simulated Annealing, and Tabu Search for the Minimum Vertex Cover problem, including all three encodings (binary, set-based, edge-centric), optimized fitness functions, benchmark instances, and result analysis scripts.

- **Enhanced Network Genetic Algorithm (ENGA)**: `https://github.com/yazid-hoblos/ENGA`

  Implementation of the enhanced networked genetic algorithm with balanced exploration and exploitation, as described in [1]. Includes the network construction framework, authority node selection, and adaptive population dynamics.

Both repositories contain detailed documentation, usage examples, and reproduction instructions for all experiments presented in this report.

B Supplementary Figures

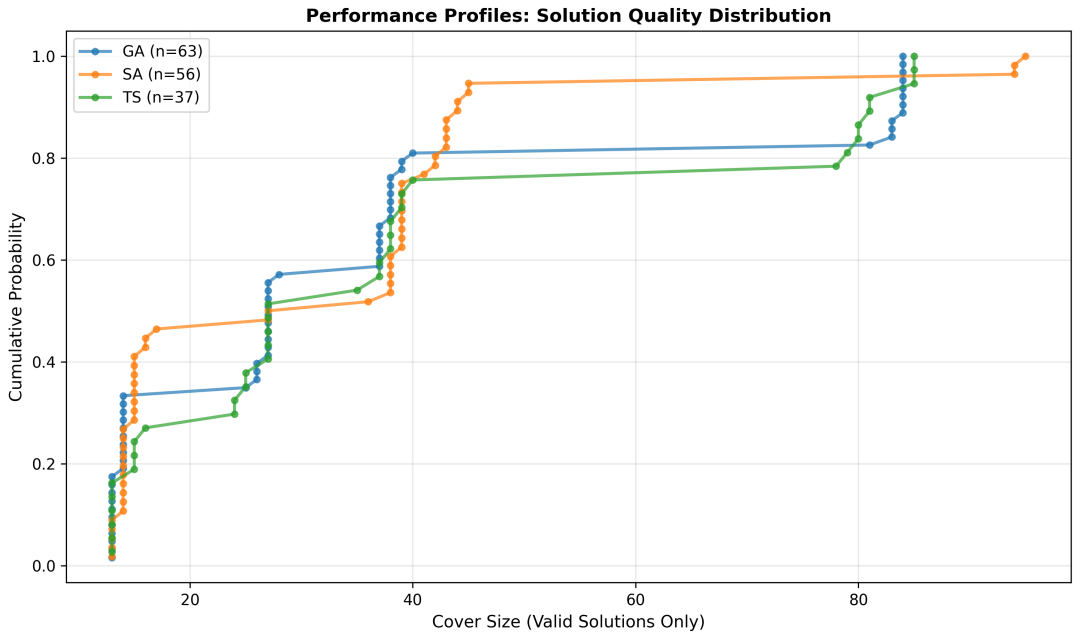

Figure 17: Performance profiles: cumulative distribution of solution quality across algorithms.

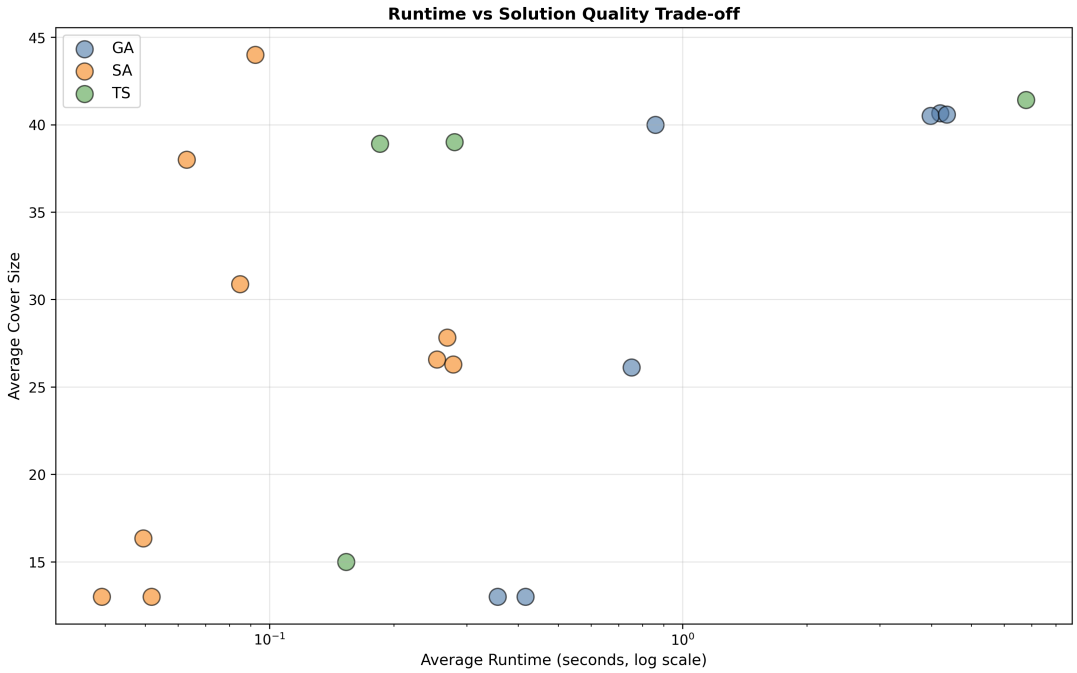

Figure 18: Runtime vs solution quality trade-off: Pareto front analysis of algorithm performance.

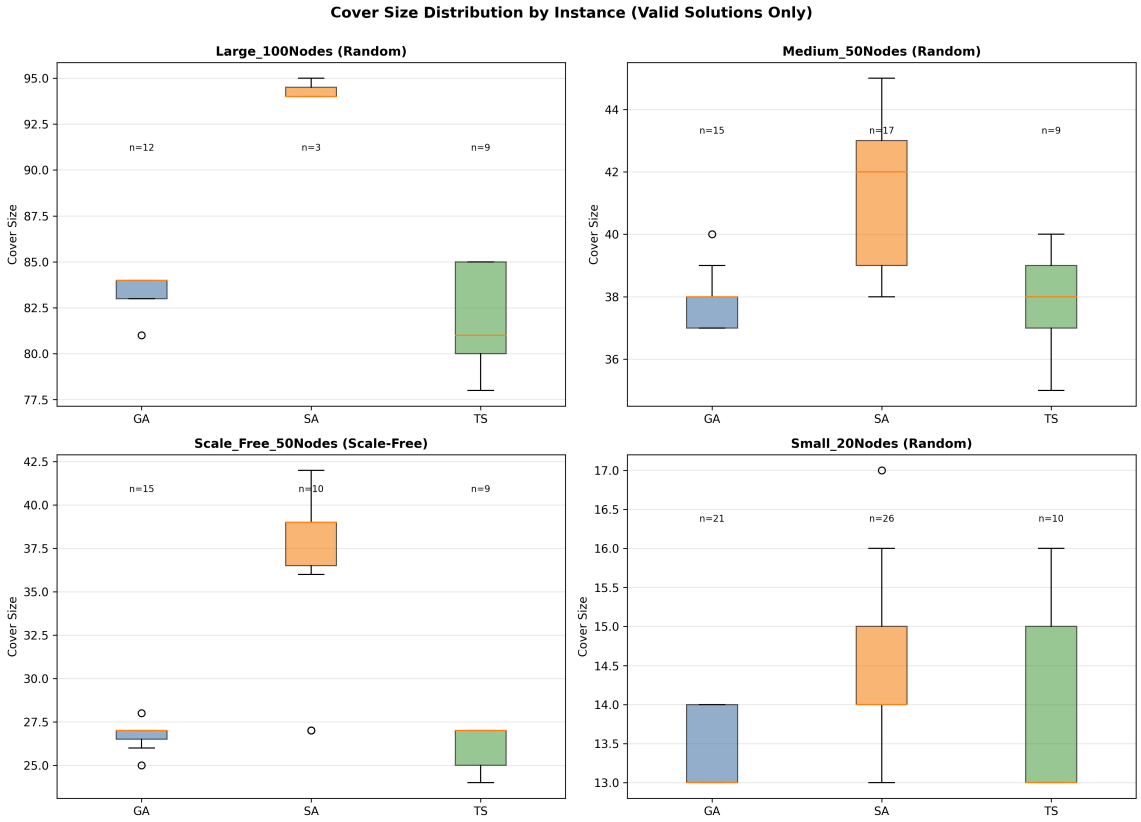

Figure 19: Baseline fitness: stratified quality by instance (average cover size and valid counts).

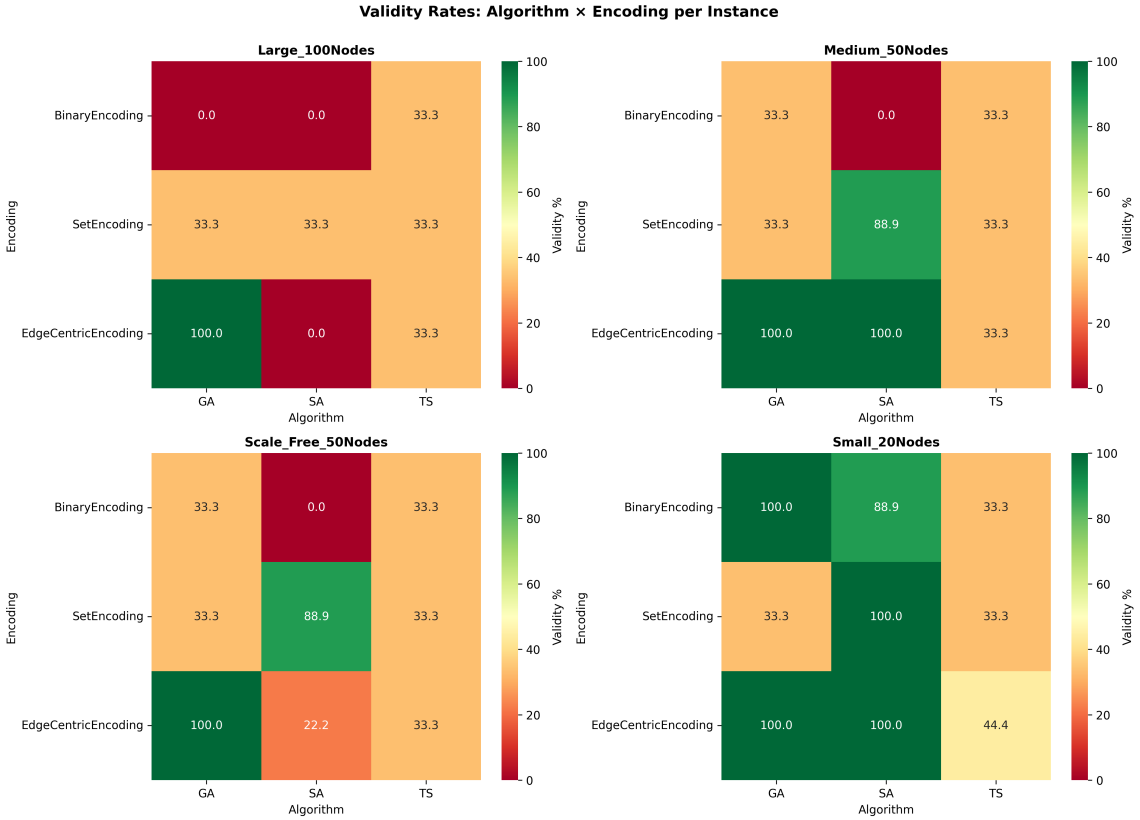

Figure 20: Baseline fitness: validity heatmap by instance and algorithm.

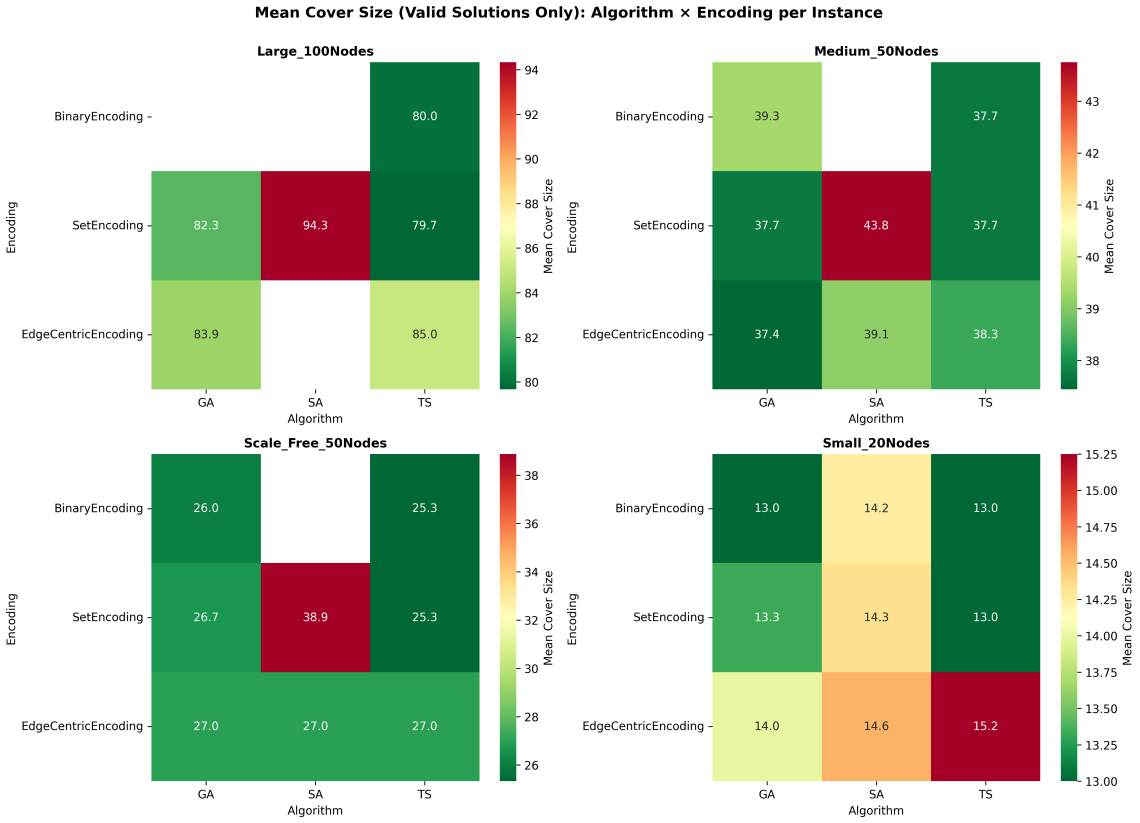

Figure 21: Baseline fitness: cover size heatmap by instance and algorithm.

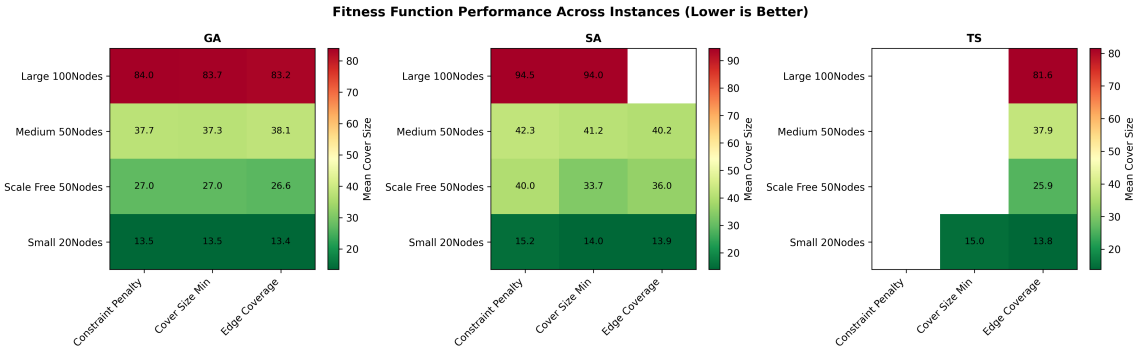

Figure 22: Baseline fitness: fitness function vs instance heatmap (mean cover size and sample counts).