Appendix: Minimum Vertex Cover Optimization Pseudocode and Implementation Details

January 28, 2026

1 Algorithm Pseudocode

1.1 Genetic Algorithm

[H] Genetic Algorithm for Minimum Vertex Cover [1] GeneticAlgorithmproblem, encoding, fitness, params population $\leftarrow$ InitializePopulation(params.population_size, encoding) best_solution $\leftarrow$ None best_fitness $\leftarrow -\infty$

generation $\leftarrow$ 1 to params.generations fitness_scores $\leftarrow$ EvaluatePopulation(population, fitness)

gen_best $\leftarrow$ ArgMax(fitness_scores) fitness_scores[gen_best] $>$ best_fitness best_fitness $\leftarrow$ fitness_scores[gen_best] best_solution $\leftarrow$ DeepCopy(population[gen_best])

elites, elite_fitness $\leftarrow$ SelectElites(population, fitness_scores, params.elitism_rate) new_population $\leftarrow$ elites

Length(new_population) $<$ params.population_size parent1 $\leftarrow$ Selection(population, fitness_scores, params.selection_type) parent2 $\leftarrow$ Selection(population, fitness_scores, params.selection_type)

Random() $<$ params.crossover_rate child1, child2 $\leftarrow$ Crossover(parent1, parent2, encoding) child1 $\leftarrow$ DeepCopy(parent1) child2 $\leftarrow$ DeepCopy(parent2)

Mutate(child1, params.mutation_rate, encoding) Mutate(child2, params.mutation_rate, encoding)

new_population.Append(child1) Length(new_population) $<$ params.population_size new_population.Append(child2)

population $\leftarrow$ new_population[0:params.population_size]

cover $\leftarrow$ encoding.SolutionToCover(best_solution) {best_solution, best_fitness, cover, is_valid}

1.2 Simulated Annealing

[H] Simulated Annealing for Minimum Vertex Cover [1] SimulatedAnnealingproblem, encoding, fitness, params current $\leftarrow$ RandomSolution(encoding) current_fitness $\leftarrow$ Evaluate(current, fitness, encoding) best $\leftarrow$ DeepCopy(current) best_fitness $\leftarrow$ current_fitness

temperature $\leftarrow$ params.initial_temperature iteration $\leftarrow$ 0

temperature $>$ params.min_temperature AND iteration $<$ params.max_iterations i $\leftarrow$ 1 to params.iterations_per_temperature neighbor $\leftarrow$ GetNeighbor(current, encoding) neighbor_fitness $\leftarrow$ Evaluate(neighbor, fitness, encoding) accept_prob $\leftarrow$ AcceptanceProbability(current_fitness, neighbor_fitness, temperature)

Random() $<$ accept_prob current $\leftarrow$ neighbor current_fitness $\leftarrow$ neighbor_fitness

current_fitness $>$ best_fitness best $\leftarrow$ DeepCopy(current) best_fitness $\leftarrow$ current_fitness

iteration $\leftarrow$ iteration $+$ 1 iteration $\geq$ params.max_iterations **break**

temperature $\leftarrow$ temperature $\times$ params.cooling_rate

cover $\leftarrow$ encoding.SolutionToCover(best) {best, best_fitness, cover, is_valid}

AcceptanceProbabilitycurrent_fitness, neighbor_fitness, temperature neighbor_fitness $\geq$ current_fitness 1.0 exp $\left(\frac{\text{neighbor\_fitness}-\text{current\_fitness}}{\text{temperature}}\right)$

1.3 Tabu Search

[H] Tabu Search for Minimum Vertex Cover [1] TabuSearchproblem, encoding, fitness, params current $\leftarrow$ RandomSolution(encoding) current_fitness $\leftarrow$ Evaluate(current, fitness, encoding) best $\leftarrow$ DeepCopy(current) best_fitness $\leftarrow$ current_fitness

tabu_list $\leftarrow$ EmptyQueue(max_size=params.tabu_list_size) tabu_list.Add(Hash(current))

iteration $\leftarrow$ 0 no_improvement_count $\leftarrow$ 0

iteration $<$ params.max_iterations AND no_improvement_count $<$ 100 neighbors $\leftarrow$ GenerateNeighborhood(current, encoding)

neighbors is empty **break**

best_neighbor $\leftarrow$ None best_neighbor_fitness $\leftarrow$ $-\infty$

each neighbor **in** neighbors neighbor_hash $\leftarrow$ Hash(neighbor) is_tabu $\leftarrow$ tabu_list.Contains(neighb
neighbor_fitness $\leftarrow$ Evaluate(neighbor, fitness, encoding)

aspiration_met $\leftarrow$ (params.aspiration AND neighbor_fitness $>$ best_fitness)

(NOT is_tabu AND neighbor_fitness $>$ best_neighbor_fitness) OR
best_neighbor $\leftarrow$ neighbor best_neighbor_fitness $\leftarrow$ neighbor_fitness

best_neighbor is None **break**

current $\leftarrow$ best_neighbor current_fitness $\leftarrow$ best_neighbor_fitness tabu_list.Add(Hash(current))

current_fitness $>$ best_fitness best $\leftarrow$ DeepCopy(current) best_fitness $\leftarrow$ current_fitness
no_improvement_count $\leftarrow$ 0 no_improvement_count $\leftarrow$ no_improvement_count $+$ 1

iteration $\leftarrow$ iteration $+$ 1

cover $\leftarrow$ encoding.SolutionToCover(best) {best, best_fitness, cover, is_valid}

2 Encoding Implementation

2.1 Binary Encoding Example

```python
def binary_solution_to_cover(solution, num_nodes):
    """Convert binary vector to vertex cover."""
    cover = set()
    for i in range(num_nodes):
        if solution[i] == 1:
            cover.add(i)
    return cover

def binary_cover_to_solution(cover, num_nodes):
    """Convert cover to binary vector."""
    solution = [0] * num_nodes
    for node in cover:
```

```python
        solution[node] = 1
    return solution

def binary_random_solution(num_nodes):
    """Generate random binary vector."""
    return [random.randint(0, 1) for _ in range(num_nodes)]

def binary_mutation(solution, mutation_rate, num_nodes):
    """Bit-flip mutation."""
    for i in range(num_nodes):
        if random.random() < mutation_rate:
            solution[i] = 1 - solution[i]

def binary_crossover(parent1, parent2):
    """Uniform crossover for binary vectors."""
    child1, child2 = [], []
    for i in range(len(parent1)):
        if random.random() < 0.5:
            child1.append(parent1[i])
            child2.append(parent2[i])
        else:
            child1.append(parent2[i])
            child2.append(parent1[i])
    return child1, child2
```

Listing 1: Binary Encoding - Solution to Cover Conversion

2.2 Set-Based Encoding Example

```python
def set_solution_to_cover(solution):
    """Convert sorted list to cover set."""
    return set(solution)

def set_cover_to_solution(cover):
    """Convert cover to sorted list."""
    return sorted(list(cover))

def set_random_solution(num_nodes):
    """Generate random set of nodes."""
    num_selected = random.randint(1, max(2, num_nodes // 2))
    nodes = random.sample(range(num_nodes), num_selected)
    return sorted(nodes)

def set_mutation(solution, mutation_rate, num_nodes):
    """Add/remove nodes from set."""
    if random.random() < mutation_rate / 2 and len(solution) > 0:
        # Remove random node
        idx = random.randint(0, len(solution) - 1)
        solution.pop(idx)

    if random.random() < mutation_rate / 2:
        # Add random node not in cover
        node = random.randint(0, num_nodes - 1)
        if node not in solution:
            solution.append(node)
            solution.sort()

```

```python
def set_crossover(parent1, parent2):
    """Single-point crossover for variable-length sets."""
    # For variable-length, perform standard single-point split
    # and recombine (alternative: intersection/union-based)
    if len(parent1) <= 1:
        return list(parent1), list(parent2)

    point = random.randint(1, len(parent1) - 1)
    child1 = parent1[:point] + parent2[point:]
    child2 = parent2[:point] + parent1[point:]
    return child1, child2
```

Listing 2: Set-Based Encoding - Solution to Cover Conversion

3 Fitness Function Implementation

```python
def cover_size_minimization(cover, problem):
    """Fitness = 1 / (1 + normalized_size)"""
    size = len(cover)
    normalized_size = size / problem.num_nodes
    return 1.0 / (1.0 + normalized_size)

def constraint_penalty(cover, problem, penalty_weight=1.0):
    """
    Fitness with constraint penalty.
    Valid covers get positive fitness; invalid get penalized.
    """
    is_valid = problem.is_valid_cover(cover)
    uncovered = count_uncovered_edges(cover, problem)
    cover_size = len(cover)

    if is_valid:
        return 1.0 - (cover_size / problem.num_nodes) * 0.5
    else:
        penalty = penalty_weight * (uncovered + cover_size / problem.
            num_nodes)
        return max(0.0, 1.0 - penalty)

def edge_coverage_optimization(cover, problem, size_weight=0.3):
    """
    Multi-objective: maximize edge coverage, minimize cover size.
    Fitness = (covered_edges / total) - weight * (size / nodes)
    """
    covered = count_covered_edges(cover, problem)
    coverage_ratio = covered / problem.num_edges if problem.num_edges >
        0 else 0

    size_penalty = (len(cover) / problem.num_nodes) * size_weight

    fitness = coverage_ratio - size_penalty
    return max(0.0, fitness)

def count_uncovered_edges(cover, problem):
    """Count edges not covered by solution."""
    uncovered = 0
    for u, v in problem.edges:
```

```
        if u not in cover and v not in cover:
            uncovered += 1
    return uncovered

def count_covered_edges(cover, problem):
    """Count edges covered by solution."""
    covered = 0
    for u, v in problem.edges:
        if u in cover or v in cover:
            covered += 1
    return covered
```

Listing 3: Fitness Function Examples

4 Experimental Framework

```
def run_experiments(num_runs=5):
    """Run full experimental suite."""
    instances = generate_benchmark_instances()
    results = []

    for problem, instance_name in instances:
        print(f"Processing {instance_name}...")

        encodings = [
            BinaryEncoding(),
            SetEncoding(),
            EdgeCentricEncoding(problem.edges)
        ]

        fitness_functions = [
            CoverSizeMinimization(problem),
            ConstraintPenalty(problem, penalty_weight=1.0),
            EdgeCoverageOptimization(problem, size_weight=0.3)
        ]

        for run_id in range(num_runs):
            for encoding in encodings:
                for fitness_func in fitness_functions:
                    # Genetic Algorithm
                    ga = GeneticAlgorithm(
                        problem, encoding, fitness_func,
                        params=GAParams(population_size=100,
                            generations=300)
                    )
                    result_ga = ga.run()
                    results.append({
                        'instance': instance_name,
                        'algorithm': 'GA',
                        'encoding': encoding.get_name(),
                        'fitness_func': fitness_func.get_name(),
                        'run': run_id,
                        'cover_size': result_ga['best_cover_size'],
                        'is_valid': result_ga['is_valid'],
                        'fitness': result_ga['best_fitness']
                    })
```

```python

                        # Simulated Annealing
                        sa = SimulatedAnnealing(
                            problem, encoding, fitness_func,
                            params=SAParams(initial_temperature=100.0,
                                max_iterations=5000)
                        )
                        result_sa = sa.run()
                        results.append({...})  # Similar structure

                        # Tabu Search
                        ts = TabuSearch(
                            problem, encoding, fitness_func,
                            params=TSParams(tabu_list_size=50,
                                max_iterations=5000)
                        )
                        result_ts = ts.run()
                        results.append({...})  # Similar structure

    # Save and analyze results
    save_to_csv(results, 'results.csv')
    print_summary_statistics(results)
    return results
```

Listing 4: Main Experiment Loop Structure

5 Instance Generation

```python
import networkx as nx
import random

def generate_erdos_renyi_instance(num_nodes, edge_probability, seed=
    None):
    """Generate random graph using ER model."""
    if seed:
        random.seed(seed)

    graph = nx.erdos_renyi_graph(num_nodes, edge_probability)
    return MinimumVertexCoverProblem(graph)

def generate_scale_free_instance(num_nodes, m=2, seed=None):
    """Generate scale-free graph (Barabasi-Albert model)."""
    if seed:
        random.seed(seed)

    graph = nx.barabasi_albert_graph(num_nodes, m)
    return MinimumVertexCoverProblem(graph)

def generate_benchmark_suite():
    """Generate standard benchmark instances."""
    instances = []

    # Small ER instance
    instances.append((
        generate_erdos_renyi_instance(20, 0.3, seed=42),
        "small_20nodes"
```

```
    ))

    # Medium ER instance
    instances.append((
        generate_erdos_renyi_instance(50, 0.25, seed=43),
        "medium_50nodes"
    ))

    # Large ER instance
    instances.append((
        generate_erdos_renyi_instance(100, 0.15, seed=44),
        "large_100nodes"
    ))

    # Scale-free instance
    instances.append((
        generate_scale_free_instance(50, m=3, seed=45),
        "scale_free_50nodes"
    ))

    return instances
```

Listing 5: Benchmark Instance Generation

6 Key Implementation Details

6.1 Parameter Selection Rationale

1. **GA Population Size (100)**: Balances diversity against computational cost. Too small (20-30) causes premature convergence; too large (500+) slows iterations.

2. **GA Generations (300)**: Approximately 30,000 fitness evaluations. Provides reasonable runtime while avoiding convergence plateau.

3. **SA Initial Temperature (100.0)**: Calibrated so that $\exp(-100/100) = \exp(-1) \approx 0.37$, giving 37% acceptance of random moves initially.

4. **SA Cooling Rate (0.95)**: Geometric schedule slower than 0.9 (too aggressive) but faster than 0.99 (too slow). Reaches min temperature in 90 iterations.

5. **TS Tabu List Size (50)**: Approximately 10% of solution space (for 20-100 node graphs). Prevents recent cycling without over-constraining.

6.2 Reproducibility

All experiments use fixed random seeds:

- Instance generation: seeds 42-45
- Algorithm runs: Python random.seed() and numpy.random.seed() set before each run
- Enables exact replication of reported results