Critical Analysis: Quantum Speedup Claims in Coalition Formation

Technical Report on Mohseni et al. (2024) vs. Rotation Optimization Benchmarks

OQI-UC002-DWave Project
Technical Analysis Report

December 10, 2025

Abstract

This report provides a comprehensive technical analysis of the methodology used by Mohseni et al. (arXiv:2405.11917) to demonstrate "quantum scaling advantage" in approximate optimization for coalition formation with 100+ agents. We identify **five critical methodological differences** that explain their claimed 100% solution quality and favorable runtime scaling compared to classical solvers, contrasting with our rotation optimization benchmarks where direct QPU shows 87% optimality gap. Our analysis reveals that their approach uses problem decomposition, specialized hardware (DWaveCliqueSampler), and fundamentally different problem characteristics that dramatically reduce embedding overhead. **Key finding: The speedup is real but highly problem-specific and relies on decomposition + clique embedding, not raw QPU superiority.**

Contents

1 Executive Summary

1.1 The Fundamental Question

Why does Mohseni et al. achieve 100% solution quality with D-Wave while our rotation benchmarks show 87% optimality gap?

1.2 Key Findings

1. **Problem Decomposition**: They solve many *small* ($n \leq 20$ variables) independent QUBOs, not one large problem
2. **Clique Embedding**: Uses `DWaveCliqueSampler` which exploits hardware cliques (16-20 qubits) for *zero* embedding overhead
3. **Problem Structure**: Coalition splitting creates *sparse, balanced* graph bisection problems vs. our *dense, frustrated* rotation coupling
4. **Iterative Refinement**: Hierarchical decomposition allows classical+quantum hybrid approach across levels
5. **Apples vs. Oranges**: Benchmarking "approximate optimization" where *any* feasible solution with value $\geq$ threshold is acceptable

Verdict: The speedup is legitimate but contingent on specific problem properties. Not generalizable to arbitrary combinatorial optimization.

2 Problem Formulation Comparison

2.1 Our Problem: Multi-Period Rotation Optimization

2.1.1 Mathematical Formulation

$$\max \sum_{f,c,t} B_c L_f Y_{f,c,t} + \gamma \sum_{f,c,c',t} R_{c,c'} L_f Y_{f,c,t-1} Y_{f,c',t} + \text{spatial} + \text{penalties} \tag{1}$$

Characteristics:

- **Variables**: 90-900 binary variables ($5 \times 6 \times 3$ to $50 \times 6 \times 3$)
- **Coupling**: Dense temporal + spatial quadratic terms (1860 interactions for 90 vars)
- **Frustration**: 86% negative synergies (spin-glass structure)
- **Constraints**: Soft penalties in objective (CQM $\rightarrow$ BQM conversion)
- **Embedding Overhead**: 90 logical $\rightarrow$ 651 physical qubits (**7.2**$\times$)

2.2 Their Problem: Coalition Structure Generation (CSG)

2.2.1 Mathematical Formulation

Coalition splitting via graph bisection:

$$\min_{x \in \{0,1\}^n} \sum_{i<j} w_{ij}(x_i x_i + x_j x_j - 2x_i x_j) \tag{2}$$

where w_{ij} = edge weight between agents i, j in coalition graph.

Characteristics:

- **Variables**: 5-20 binary variables per subproblem (*never* full $n = 100+!$)
- **Coupling**: Graph bisection (sparse, balanced cuts preferred)
- **Frustration**: Moderate (not designed to be frustrated)
- **Constraints**: None (unconstrained QUBO)
- **Embedding Overhead**: **Zero** (fits in hardware cliques)

2.2.2 Critical Insight: Hierarchical Decomposition

Algorithm 1 Coalition Formation Algorithm (Mohseni et al.)

```
1: Start with $\mathcal{C} = \{[0, 1, \ldots, N-1]\}$ (all agents in one coalition)
2: for iteration = 1 to $N$ do
3:   $\mathcal{C}_{\text{new}} \leftarrow \mathcal{C}$
4:   for each coalition $c \in \mathcal{C}$ with $|c| > 1$ do
5:     // Build QUBO for splitting $c$ into $c_1, c_2$
6:     $Q \leftarrow \text{BuildBisectionQUBO}(c)$    $\leftarrow$ Only $|c|$ variables!
7:     $(c_1, c_2) \leftarrow \text{QPU.Solve}(Q)$    $\leftarrow$ Small problem!
8:     if $V(c_1) + V(c_2) > V(c)$ then
9:       $\mathcal{C}_{\text{new}} \leftarrow \mathcal{C}_{\text{new}} \setminus \{c\} \cup \{c_1, c_2\}$
10:    end if
11:  end for
12:  if $|\mathcal{C}| == |\mathcal{C}_{\text{new}}|$ then
13:    break    // No improvement, converged
14:  end if
15:  $\mathcal{C} \leftarrow \mathcal{C}_{\text{new}}$
16: end for
17: return $\mathcal{C}$
```

Key Observations:

1. QPU *never* sees a problem with 100+ variables
2. Maximum subproblem size ≈ 20 variables (fits hardware cliques perfectly)
3. Hundreds of small independent solves vs. one large monolithic solve
4. Classical orchestration + quantum subroutines

3 Critical Methodological Differences

3.1 Difference 1: Embedding Strategy

3.1.1 What is DWaveCliqueSampler?

The D-Wave Pegasus topology contains hardware *cliques* of size 16-20 qubits that are **fully connected** (all-to-all coupling). `DWaveCliqueSampler` automatically:

- Detects if problem fits in a clique ($n \leq 16$)
- Maps directly to hardware qubits (bijection, no chains)
- Eliminates embedding overhead entirely

Aspect	Our Approach	Mohseni et al.
Sampler	`DWaveSampler + EmbeddingComposite`	`DWaveCliqueSampler`
Logical Variables	90-900	5-20 per solve
Physical Qubits	651 (7.2× overhead)	16-20 (1.0× overhead)
Embedding Time	43-80 seconds	Negligible ($<$ 1s)
Max Chain Length	9	1 (no chains!)
Chain Breaks	0.02-0.04%	0% (cliques are fully connected)

Table 1: Embedding Comparison

Code Evidence:

```
# From Main.ipynb line 78
sampler = DWaveCliqueSampler()
coalitions3 = Dwave.solve(value_agent, edges, timeout[num][idx],
                          Dwave_inf, num, idx, rest_inf_dwave, sampler)

# From utils.py line 55
def solve_with_dwave(Q, n, Dwave_inf, num, idx):
    bqm = dimod.BinaryQuadraticModel.from_qubo(Q)
    S = sampler.sample(bqm, num_reads=100)  # Direct clique embedding!
    answer = S.lowest().samples()[0]
    solution = [answer[i] for i in range(n)]
    return solution
```

This is the smoking gun: Problems with $n \leq 16$ have *perfect* embeddings. Our 90-variable problem requires complex chain-based embeddings with 7× overhead.

3.2 Difference 2: Problem Size Distribution

Metric	Our Rotation Problem	Their Coalition Splits
Initial Problem Size	90-900 vars	100 agents (full graph)
Subproblem Size	N/A (monolithic)	5-20 vars per split
Number of QPU Calls	1	∼100-300 (iterative)
Total QPU Access Time	0.034s (1 solve)	∼10-30s (many solves)
Embedding per Solve	43-80s (hard)	$<$ 0.1s (clique)
Total Wall Time	47s (embed + QPU)	40-60s (many fast solves)

Table 2: Problem Scale Comparison

Analysis: Their total QPU time is *higher* than ours, but amortized over hundreds of *trivial* embeddings. Our approach spends 95% of time on one giant embedding.

3.3 Difference 3: Problem Hardness

3.3.1 Frustration and Landscape Structure

Our Problem (Rotation):

- 86% negative synergies → frustrated spin glass
- Deep local minima (Gurobi takes 120s, 77K branch-and-bound nodes)
- Integrality gap > 700%

- QPU finds solution with 87% optimality gap (trapped in local minima)

Their Problem (Graph Bisection):

- Balanced cut objective (not highly frustrated)
- Any partition with value above threshold is acceptable ("approximate optimization")
- Problem structure ensures *many* good solutions exist
- QPU samples multiple solutions, picks best among them

3.3.2 What is "100% Solution Quality"?

From the paper abstract: *"quantum annealing on DWave can achieve solutions of comparable quality to our best classical solver."*

Key phrase: "comparable quality" $\neq$ "optimal."

Their benchmark is *not* comparing to Gurobi's exact optimum, but to:

- Tabu search (heuristic)
- Simulated annealing (heuristic)
- QBSolv (D-Wave's own classical decomposition + SA)

Translation: "100% solution quality" means QPU matches the heuristic solvers, not the global optimum. For easy graph bisection problems, heuristics often find near-optimal solutions, so this comparison is favorable.

3.4 Difference 4: Constraint Handling

Aspect	**Our Approach**	**Mohseni et al.**
Constraint Type	Hard (`CQM`)	None (unconstrained QUBO)
CQM $\rightarrow$ BQM Conversion	Yes (penalty method)	N/A
Lagrange Multipliers	50.0 (tuned)	N/A
Variable Expansion	90 $\rightarrow$ 120 BQM vars	No expansion
Feasibility Risk	High (penalties may fail)	None (all solutions valid)

Table 3: Constraint Handling Comparison

Impact: Our CQM $\rightarrow$ BQM conversion adds:

- Extra variables (slack variables for inequalities)
- Penalty terms that compete with objective
- Risk of constraint violations if penalties too weak
- Risk of objective suppression if penalties too strong

Their unconstrained QUBO has *none* of these issues.

3.5 Difference 5: Benchmarking Philosophy

3.5.1 Their Definition: Approximate Optimization

From the paper: *"Approximate optimization is particularly critical for industrial use cases requiring real-time optimization, where finding high-quality solutions quickly is often more valuable than achieving exact solutions more slowly."*

Benchmark Criteria:

- Find *good* solution within time budget
- "Good" = within 5-10% of best known (not necessarily optimal)
- Time-to-solution (wall clock) is the primary metric
- Solution quality is *secondary* (as long as "good enough")

3.5.2 Our Implicit Definition: Exact Optimization

Benchmark Criteria:

- Compare to Gurobi's *optimal* solution (ground truth)
- Report optimality gap (our solution vs. true optimum)
- Measure: 87% gap means we found 13% of optimal objective
- Time is secondary (we measure both embedding + solving)

Implication: We are solving a *harder* problem (exact optimization of highly frustrated landscape) while they solve an *easier* problem (approximate optimization of moderate landscape).

4 Code-Level Analysis

4.1 Their Solving Pipeline

```
# Step 1: Build QUBO for coalition split (small!)
def split(c, edges, timeout, Dwave_inf, num, idx):
    Q = {}
    for i in range(len(c)):  # len(c) = 5-20 typically
        for j in range(len(c)):
            if i < j:
                utils.add(Q, i, i, edges[(c[i], c[j])])
                utils.add(Q, j, j, edges[(c[i], c[j])])
                utils.add(Q, i, j, -2*edges[(c[i], c[j])])

    # Step 2: Solve with DWaveCliqueSampler (fits in clique!)
    solution = utils.solve_with_dwave(Q, len(c), Dwave_inf, num, idx)

    # Step 3: Split coalition based on solution
    c1 = [c[k] for k in range(len(c)) if solution[k] == 1]
    c2 = [c[k] for k in range(len(c)) if solution[k] == 0]
    return c1, c2

# Clique sampler call (no embedding needed!)
def solve_with_dwave(Q, n, Dwave_inf, num, idx):
    bqm = dimod.BinaryQuadraticModel.from_qubo(Q)
    S = sampler.sample(bqm, num_reads=100)  # sampler = DWaveCliqueSampler
    answer = S.lowest().samples()[0]
    solution = [answer[i] for i in range(n)]
    return solution
```
Listing 1: Mohseni et al. Solving Pipeline

Key Observations:

1. QUBO size $n \times n$ where $n = |c| \leq 20$
2. Max $20 \times 20 = 400$ entries, but sparse (only edges present)
3. `DWaveCliqueSampler` handles $n \leq 16$ natively
4. For $n = 17 - 20$, uses minimal chains (max chain length 2-3)
5. No CQM, no constraint conversion, direct QUBO $\rightarrow$ QPU

4.2 Our Solving Pipeline

```
# Step 1: Build CQM with constraints (large, coupled)
def build_rotation_cqm(data, n_periods=3):
    cqm = ConstrainedQuadraticModel()
    Y = {}
    for f in farm_names:  # 5-50 farms
        for c in families_list:  # 6 families
            for t in range(1, n_periods + 1):  # 3 periods
                Y[(f, c, t)] = Binary(f"Y_{f}_{c}_t{t}")

    # Objective: linear + quadratic rotation + spatial (1860 terms!)
    objective = ...  # Dense coupling
    cqm.set_objective(-objective)

    # Constraints: <= 2 crops per period per farm
    for f in farm_names:
        for t in range(1, n_periods + 1):
            cqm.add_constraint(sum(Y[(f, c, t)] for c in families_list) <= 2)

    return cqm  # 90 vars, 15 constraints

# Step 2: Convert CQM to BQM (penalty method, adds variables)
bqm, info = cqm_to_bqm(cqm, lagrange_multiplier=50.0)
# Result: 90 logical -> 120 BQM variables, 1860 quadratic terms

# Step 3: Find embedding (expensive!)
embedding = find_embedding(source, target, timeout=200)
# Result: 120 BQM vars -> 651 physical qubits (takes 43-80s)

# Step 4: Sample on QPU
sampler = FixedEmbeddingComposite(qpu, embedding)
sampleset = sampler.sample(bqm, num_reads=1000)
# Result: 0.034s QPU time, but 87% optimality gap (trapped in local minima)
```
Listing 2: Our Rotation Solving Pipeline

Key Differences:

- CQM with constraints (they have none)
- 90 logical vars vs. their ≤ 20
- Penalty conversion increases to 120 BQM vars
- Embedding 120 vars to 651 qubits ($7\times$ overhead)

- Dense coupling (1860 interactions) vs. their sparse bisection
- 86% frustration vs. their balanced cuts

5 Why the Speedup is Real (But Limited)

5.1 Legitimate Advantages

1. Clique Embedding Eliminates Overhead

- For $n \leq 16$, embedding is *free* (direct mapping)
- No chains $\rightarrow$ no chain breaks $\rightarrow$ no error accumulation
- Scales to hundreds of independent small solves efficiently

2. Decomposition Matches Hardware Constraints

- Coalition formation is *naturally* hierarchical
- Each subproblem is independent (parallel QPU calls)
- Problem structure "fits" the hardware topology

3. Approximate Optimization Lowers Bar

- Don't need exact optimum, just "good enough"
- Quantum annealing finds many diverse solutions quickly
- Classical refinement can improve QPU output

5.2 Limitations and Non-Generalizability

1. Problem Size Limitation

- Only works for problems that decompose into $n \leq 16$ subproblems
- For $n > 20$, embedding overhead reappears
- Cannot handle monolithic problems (like our rotation)

2. Problem Structure Requirement

- Requires natural decomposition strategy (not always available)
- Graph bisection is "easy" for quantum annealers (balanced cuts)
- Frustrated, dense problems don't decompose well

3. Comparison Bias

- Benchmarked against heuristics (Tabu, SA), not exact solvers
- Gurobi mentioned but not used as ground truth
- "100% solution quality" relative to other heuristics, not optimum

6 Apples-to-Apples Comparison: What If We Used Their Method?

6.1 Hypothetical: Decompose Rotation Problem

Could we decompose our rotation problem to fit cliques?

Option 1: Decompose by Farm

- Each farm: 6 families $\times$ 3 periods = 18 variables
- Fits in clique! ($n = 18 \leq 20$)
- Problem: Rotation synergies couple farms *temporally* (not just spatially)
- Problem: Spatial coupling between farms requires coordination

Option 2: Decompose by Period

- Each period: 5 farms $\times$ 6 families = 30 variables
- Too large for clique! ($n = 30 > 20$)
- Problem: Rotation synergies couple periods (can't separate)

Verdict: Our problem has *dense global coupling* that resists decomposition. Their problem has *local structure* amenable to hierarchical splitting.

6.2 Hypothetical: Solve Coalition Formation with Our Method

What if they used `EmbeddingComposite` instead of `DWaveCliqueSampler`?

- For $n = 20$ variables: ~50-100 physical qubits, embedding time ~1-5s
- Still manageable, but 10-50× slower embedding per solve
- Over 300 solves: 300-1500s embedding overhead (vs. their <10s)
- Speedup disappears! Classical would win.

Conclusion: Their speedup is *contingent* on clique embedding availability.

7 Recommendations for Our Project

7.1 What We Should *Not* Do

1. **Don't expect direct QPU to match Gurobi on rotation:** Our 87% gap is realistic for this problem class
2. **Don't use `EmbeddingComposite` for large monolithic problems:** Embedding overhead destroys quantum advantage
3. **Don't compare approximate to exact optimization:** Different goals → different metrics

7.2 What We *Should* Do

1. **Implement Hierarchical Decomposition:**
   - Decompose rotation by period (solve periods sequentially)
   - Use classical coordination between periods
   - Decompose spatial coupling via Louvain/Spectral clustering
   - Target subproblem size ≤ 20 variables
2. **Explore `DWaveCliqueSampler`:**
   - Redesign formulation to fit cliques ($n \leq 16$)
   - Use iterative refinement (solve small subproblems repeatedly)
   - Accept approximate solutions (don't aim for exact optimum)
3. **Use Hybrid Solvers Properly:**
   - `LeapHybridCQMSampler` for full rotation problem
   - Let D-Wave's classical decomposition handle orchestration
   - Focus on time-to-good-solution, not time-to-optimal
4. **Benchmark Fairly:**
   - If using approximate optimization, compare to heuristics (SA, Tabu)
   - If using exact optimization, compare to Gurobi
   - Report both optimality gap *and* time-to-solution
   - Be transparent about which regime we're in

8 Conclusion

8.1 Summary of Findings

Factor	**Mohseni et al. (Speedup)**	**Our Project (No Speedup)**
Problem Size	5-20 vars per subproblem	90-900 vars (monolithic)
Embedding	DWaveCliqueSampler (zero overhead)	EmbeddingComposite ($7\times$ overhead)
Problem Structure	Sparse graph bisection	Dense temporal + spatial coupling
Frustration	Low (balanced cuts)	High (86% negative synergies)
Constraints	None (unconstrained QUBO)	Hard constraints (CQM $\rightarrow$ BQM)
Decomposition	Natural hierarchical	Resists decomposition
Benchmark Goal	Approximate (match heuristics)	Exact (match Gurobi)
Solution Quality	100% (vs. heuristics)	13% (vs. optimal)

Table 4: Comprehensive Comparison

8.2 Final Verdict

Is their speedup real? **Yes**, for problems with:

1. Natural decomposition into $n \leq 16$ subproblems
2. Sparse, balanced structure (not highly frustrated)
3. No hard constraints
4. Approximate optimization acceptable

Is their speedup generalizable? **No**, because:

1. Clique embedding only works for tiny problems ($n \leq 16$)
2. Decomposition strategy is problem-specific
3. Highly coupled, frustrated problems cannot be decomposed
4. Monolithic large problems hit embedding wall

8.3 Implications for Quantum Advantage

- **Quantum advantage exists**, but in a *narrow regime*
- Success requires **algorithm-hardware co-design**
- Problem must be **reformulated** to exploit hardware topology
- Direct QPU is **not** a drop-in replacement for classical solvers
- Hybrid (classical + quantum) is the practical path forward

Key Takeaway: Mohseni et al. demonstrate speedup by *engineering the problem to fit the hardware*, not by raw quantum superiority. This is legitimate but requires careful problem selection and reformulation. Our rotation problem, as currently formulated, is fundamentally incompatible with their approach.

8.4 Recommended Next Steps

1. **Immediate:** Test hierarchical decomposition (period-by-period)
2. **Short-term:** Benchmark `LeapHybridCQMSampler` vs. Gurobi
3. **Medium-term:** Redesign rotation formulation for clique compatibility
4. **Long-term:** Develop problem-specific decomposition strategies

"The quantum advantage is not a matter of hardware speed, but of algorithm-problem-hardware alignment." — This analysis (2025)