Technical Report: Quantum Annealing Decomposition Methods for Large-Scale Crop Allocation Optimization

OQI-UC002-DWave Project

December 3, 2025

Abstract

This technical report presents a comprehensive analysis of decomposition methods for solving large-scale crop allocation problems on D-Wave quantum processing units (QPUs). We evaluate eight distinct approaches: direct QPU embedding, plot-based decomposition, multilevel partitioning, Louvain community detection, CQM-first decomposition, coordinated master-subproblem, spectral clustering, and **HybridGrid partitioning**. Each method addresses the fundamental challenge of mapping constrained quadratic models (CQMs) with hundreds of variables onto quantum hardware with limited qubit connectivity. We provide detailed mathematical formulations, algorithmic descriptions, implementation details, and comparative performance analysis across multiple problem scales. Our key finding is that **HybridGrid(5,9)** achieves only 10% optimality gap at 1000 farms while maintaining zero constraint violations, significantly outperforming traditional Multilevel decomposition (40% gap).

1 Introduction

1.1 Problem Overview

The crop allocation optimization problem seeks to assign crops to farms while maximizing nutritional and sustainability benefits subject to multiple constraints:

$$\text{maximize} \quad \sum_{f \in \mathcal{F}} \sum_{c \in \mathcal{C}} b_c \cdot a_f \cdot Y_{f,c} \tag{1}$$

$$\text{subject to} \quad \sum_{c \in \mathcal{C}} Y_{f,c} \leq 1, \quad \forall f \in \mathcal{F} \tag{2}$$

$$Y_{f,c} \leq U_c, \quad \forall f \in \mathcal{F}, c \in \mathcal{C} \tag{3}$$

$$U_c \leq \sum_{f \in \mathcal{F}} Y_{f,c}, \quad \forall c \in \mathcal{C} \tag{4}$$

$$\sum_{c \in G_g} U_c \geq m_g, \quad \forall g \in \mathcal{G} \tag{5}$$

$$\sum_{c \in G_g} U_c \leq M_g, \quad \forall g \in \mathcal{G} \tag{6}$$

where:

- $\mathcal{F}$ = set of farms (plots)
- $\mathcal{C}$ = set of crops (foods)
- $\mathcal{G}$ = set of food groups

- $Y_{f,c} \in \{0, 1\}$ = binary variable indicating if crop c is assigned to farm f
- $U_c \in \{0, 1\}$ = binary variable indicating if crop c is used anywhere
- b_c = nutritional/sustainability benefit of crop c
- a_f = area of farm f
- G_g = set of crops in food group g
- m_g, M_g = minimum and maximum number of unique crops from group g

1.2 Quantum Annealing Challenge

D-Wave quantum annealers solve quadratic unconstrained binary optimization (QUBO) problems by finding the ground state of an Ising Hamiltonian. The primary challenges are:

1. **Constraint Encoding**: CQM constraints must be converted to QUBO penalty terms with appropriate Lagrange multipliers.
2. **Limited Connectivity**: The QPU's Pegasus topology has limited qubit connectivity, requiring minor embedding that can exponentially increase physical qubit requirements.
3. **Chain Breaks**: Embedded logical qubits use chains of physical qubits; disagreement within chains causes errors.
4. **Scalability**: Direct embedding becomes infeasible beyond ∼300-500 logical variables.

2 Decomposition Methods

2.1 Method 1: Direct QPU Embedding

2.1.1 Description

Direct QPU embedding attempts to map the entire problem onto the quantum hardware without decomposition. This serves as the baseline quantum approach.

2.1.2 Algorithm

Algorithm 1 Direct QPU Embedding

Require: CQM with variables $\mathcal{V}$, constraints $\mathcal{C}$
Ensure: Solution $\mathbf{x}$ or failure
Convert CQM to BQM: BQM $\leftarrow$ cqm_to_bqm(CQM, λ)
Build source graph $G_s = (\mathcal{V}, E_s)$ from BQM quadratic terms
Get QPU target graph $G_t = (Q, E_t)$ from Pegasus topology
Find embedding: $\phi : \mathcal{V} \to 2^Q$ using minorminer
if embedding found within timeout **then**
    Sample: $\mathbf{x} \leftarrow$ QPU.sample(BQM, ϕ, n_{reads})
    return best feasible solution
else
    return FAIL (problem too large)
end if

2.1.3 Complexity

- **Time**: $O(T_{\text{embed}} + n_{\text{reads}} \cdot T_{\text{QPU}})$ where T_{embed} can be exponential
- **Space**: $O(|\mathcal{V}| \cdot c)$ physical qubits, where c is chain length (typically 2-10)
- **Limitations**: Fails when $|\mathcal{V}| > 300$-500 or high connectivity

2.2 Method 2: Plot-Based Decomposition

2.2.1 Description

Plot-based decomposition partitions the problem by farm, creating one subproblem per farm plus a master problem for unique crop tracking. This ensures constraint preservation since farms are independent.

2.2.2 Mathematical Formulation

Partition variables into farm-specific subsets plus global U variables:

$$\mathcal{P}_{\text{PlotBased}} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{|\mathcal{F}|}, \mathcal{P}_U\} \tag{7}$$

where:

- $\mathcal{P}_i = \{Y_{f_i,c} : c \in \mathcal{C}\}$ for farm f_i
- $\mathcal{P}_U = \{U_c : c \in \mathcal{C}\}$

Each farm partition has $|\mathcal{C}|$ variables (one per crop), creating $|\mathcal{F}| + 1$ partitions.

2.2.3 Algorithm

Algorithm 2 Plot-Based Decomposition

Require: Data with $|\mathcal{F}|$ farms, $|\mathcal{C}|$ crops
Ensure: Complete solution $\mathbf{x}$

Create partitions: $\mathcal{P}_f = \{Y_{f,c} : c \in \mathcal{C}\}$ for each farm f
Create U partition: $\mathcal{P}_U = \{U_c : c \in \mathcal{C}\}$
$\mathbf{x} \leftarrow \emptyset$
for each partition $\mathcal{P}$ in $\{\mathcal{P}_1, \ldots, \mathcal{P}_{|\mathcal{F}|}, \mathcal{P}_U\}$ **do**
    Build BQM for variables in $\mathcal{P}$ with objective and local constraints
    Embed BQM on QPU: $\phi_{\mathcal{P}} \leftarrow \text{find\_embedding}(\mathcal{P}, G_t)$
    Sample: $\mathbf{x}_{\mathcal{P}} \leftarrow \text{QPU.sample}(\text{BQM}_{\mathcal{P}}, \phi_{\mathcal{P}}, n_{\text{reads}})$
    Merge with conflict resolution: $\mathbf{x} \leftarrow \text{merge}(\mathbf{x}, \mathbf{x}_{\mathcal{P}})$
end for
return x

2.2.4 Conflict Resolution

When merging partition solutions, farm assignment conflicts are resolved by benefit comparison:

Algorithm 3 Conflict Resolution for Farm Assignments

Require: New assignment $Y_{f,c} = 1$, existing assignments $\mathbf{x}$

if $\exists c' : \mathbf{x}[Y_{f,c'}] = 1$ **then** ▷ Conflict detected
  $b_{\text{new}} \leftarrow b_c \cdot a_f$
  $b_{\text{old}} \leftarrow b_{c'} \cdot a_f$
  if $b_{\text{new}} > b_{\text{old}}$ **then**
    $\mathbf{x}[Y_{f,c'}] \leftarrow 0$ ▷ Replace with better option
    $\mathbf{x}[Y_{f,c}] \leftarrow 1$
  end if
else
  $\mathbf{x}[Y_{f,c}] \leftarrow 1$ ▷ No conflict
end if

2.2.5 Complexity

- **Partitions**: $|\mathcal{F}| + 1$
- **Partition size**: $|\mathcal{C}|$ variables each
- **Total QPU calls**: $|\mathcal{F}| + 1$
- **Embedding**: Fast (small partitions)
- **Constraint preservation**: Excellent (farms independent)

2.3 Method 3: Multilevel Partitioning

2.3.1 Description

Multilevel partitioning groups farms into larger clusters of size k, reducing the number of partitions at the cost of partition size and potential constraint violations.

2.3.2 Mathematical Formulation

Group farms into clusters of size k:

$$\mathcal{P}_{\text{Multilevel}} = \{\mathcal{P}_1, \ldots, \mathcal{P}_{\lceil |\mathcal{F}|/k \rceil}, \mathcal{P}_U\} \tag{8}$$

where:

$$\mathcal{P}_i = \{Y_{f,c} : f \in \mathcal{F}_i, c \in \mathcal{C}\} \tag{9}$$

and $\mathcal{F}_i$ is a subset of k farms.
Each partition has $k \cdot |\mathcal{C}|$ variables.

2.3.3 Algorithm

2.3.4 Trade-offs

- **Fewer partitions**: $\lceil |\mathcal{F}|/k \rceil + 1$ vs $|\mathcal{F}| + 1$
- **Larger partitions**: $k \cdot |\mathcal{C}|$ variables vs $|\mathcal{C}|$
- **Embedding difficulty**: Increases with k
- **Violations**: Can occur when farms in same partition compete for crops

Algorithm 4 Multilevel Partitioning (k-farm groups)

Require: Data with $|\mathcal{F}|$ farms, group size k
Ensure: Solution $\mathbf{x}$

Divide farms into groups: $\mathcal{F} = \bigcup_{i=1}^{\lceil |\mathcal{F}|/k \rceil} \mathcal{F}_i$ where $|\mathcal{F}_i| \leq k$
for each farm group $\mathcal{F}_i$ **do**
    $\mathcal{P}_i \leftarrow \{Y_{f,c} : f \in \mathcal{F}_i, c \in \mathcal{C}\}$
    Build BQM for $\mathcal{P}_i$ with one-crop constraints for each $f \in \mathcal{F}_i$
    Solve partition on QPU
    Merge solution with conflict resolution
end for
Solve U partition
return $\mathbf{x}$

2.4 Method 4: Louvain Community Detection

2.4.1 Description

Louvain decomposition uses community detection on the variable interaction graph to create partitions that minimize cross-partition edges. This is a graph-theoretic approach that adapts to problem structure.

2.4.2 Mathematical Formulation

Build interaction graph $G_{\text{int}} = (\mathcal{V}, E_{\text{int}})$ where:

$$E_{\text{int}} = \{(Y_{f,c}, Y_{f,c'}) : f \in \mathcal{F}, c \neq c' \in \mathcal{C}\} \cup \{(Y_{f,c}, U_c) : f \in \mathcal{F}, c \in \mathcal{C}\} \tag{10}$$

Apply Louvain algorithm to maximize modularity:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{11}$$

where $m = |E_{\text{int}}|$, A is adjacency matrix, k_i is degree of node i, and $\delta(c_i, c_j) = 1$ if nodes i, j are in same community.

2.4.3 Algorithm

Algorithm 5 Louvain Community Detection Decomposition

Require: Variable interaction graph G_{int}
Ensure: Partition set $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$

Initialize: each variable in its own community
repeat
    for each variable v **do**
        Find community C that maximizes modularity gain
        Move v to C if gain is positive
    end for
    Aggregate communities into super-nodes
until no modularity improvement
Split partitions exceeding size limit: $|\mathcal{P}_i| \leq \text{max\_size}$
return $\mathcal{P}$

2.4.4 Characteristics

- **Adaptive**: Partition structure follows problem connectivity
- **Many partitions**: Typically creates $|\mathcal{F}|$ to $2|\mathcal{F}|$ partitions
- **Variable partition sizes**: From 2 to max_size variables
- **Modularity optimization**: Minimizes cross-partition interactions

2.5 Method 5: CQM-First Decomposition

2.5.1 Description

CQM-first decomposition partitions at the CQM level before converting to BQM, preserving constraint structure within partitions. This addresses the fundamental issue that BQM-first approaches lose constraint information during penalty encoding.

2.5.2 Key Innovation

Standard decomposition: CQM $\rightarrow$ BQM $\rightarrow$ Partition
CQM-first: CQM $\rightarrow$ Partition $\rightarrow$ Sub-CQMs $\rightarrow$ BQMs

2.5.3 Algorithm

Algorithm 6 CQM-First Decomposition with Constraint Preservation

Require: CQM with variables $\mathcal{V}$, constraints $\mathcal{C}$
Require: Partition function $\Pi : \mathcal{V} \rightarrow \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$
Ensure: Solution $\mathbf{x}$
Partition variables: $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\} \leftarrow \Pi(\mathcal{V})$
Identify master partition $\mathcal{P}_U$ containing U variables
Phase 1: Solve Master
Extract sub-CQM for $\mathcal{P}_U$ with food group constraints
Convert to BQM: $\text{BQM}_U \leftarrow \text{cqm\_to\_bqm}(\text{Sub-CQM}_U, \lambda)$
$\mathbf{x}_U \leftarrow \text{QPU.sample}(\text{BQM}_U)$
Phase 2: Solve Subproblems with Fixed U
for partition $\mathcal{P}_i$ where $i \neq U$ **do**
    Extract sub-CQM for $\mathcal{P}_i$ with $\mathbf{x}_U$ fixed
    Convert to BQM: $\text{BQM}_i \leftarrow \text{cqm\_to\_bqm}(\text{Sub-CQM}_i, \lambda)$
    $\mathbf{x}_i \leftarrow \text{QPU.sample}(\text{BQM}_i)$
    Merge with conflict resolution: $\mathbf{x} \leftarrow \text{merge}(\mathbf{x}, \mathbf{x}_i)$
end for
return x

2.5.4 Constraint Extraction

The sub-CQM extraction process preserves constraints:

2.5.5 Advantages

- **Constraint preservation**: Constraints remain explicit within partitions
- **Better penalty encoding**: Lagrange multipliers applied per partition
- **Two-phase coordination**: Master-subproblem structure ensures global feasibility

Algorithm 7 Extract Sub-CQM

Require: CQM, partition variables $\mathcal{P}$, fixed variables $\mathcal{F}_{\text{vars}}$
Ensure: Sub-CQM containing only $\mathcal{P}$ variables

```
Create new CQM: Sub-CQM ← ∅
for constraint c ∈ CQM.constraints do
    V_c ← variables in constraint c
    V_partition ← V_c ∩ P
    V_fixed ← V_c ∩ F_vars
    if V_partition ≠ ∅ then
        Substitute fixed values into constraint
        Add simplified constraint to Sub-CQM
    end if
end for
return Sub-CQM
```

2.6 Method 6: Coordinated Master-Subproblem

2.6.1 Description

Coordinated decomposition uses a rigorous two-level optimization where the master problem selects which crops to use (U variables) and farm subproblems independently assign these crops to farms.

2.6.2 Mathematical Formulation

Master Problem:

$$\text{minimize} \quad \sum_{c \in \mathcal{C}} \lambda_c \cdot U_c \tag{12}$$

$$\text{subject to} \quad \sum_{c \in G_g} U_c \geq m_g, \quad \forall g \in \mathcal{G} \tag{13}$$

$$U_c \in \{0, 1\}, \quad \forall c \in \mathcal{C} \tag{14}$$

The master objective uses small penalties λ_c to encourage crop selection while satisfying food group diversity.

Farm Subproblems (for each farm f):

$$\text{maximize} \quad \sum_{c \in \mathcal{C}} b_c \cdot a_f \cdot Y_{f,c} \tag{15}$$

$$\text{subject to} \quad \sum_{c \in \mathcal{C}} Y_{f,c} \leq 1 \tag{16}$$

$$Y_{f,c} \leq U_c^*, \quad \forall c \in \mathcal{C} \tag{17}$$

$$Y_{f,c} \in \{0, 1\}, \quad \forall c \in \mathcal{C} \tag{18}$$

where U_c^* is the fixed value from the master solution.

2.6.3 Algorithm

2.6.4 Properties

- **Hierarchical**: Clear master-subproblem structure
- **Independent subproblems**: Farms solved in parallel

Algorithm 8 Coordinated Master-Subproblem Decomposition

Require: Data with farms $\mathcal{F}$, crops $\mathcal{C}$, food groups $\mathcal{G}$
Ensure: Solution $\mathbf{x} = (\mathbf{Y}, \mathbf{U})$
Step 1: Solve Master Problem
Build master BQM for U variables with food group constraints
$\mathbf{U}^* \leftarrow$ QPU.sample($\text{BQM}_{\text{master}}$)
selected_crops $\leftarrow \{c : U_c^* = 1\}$
Step 2: Solve Farm Subproblems
for each farm $f \in \mathcal{F}$ **do**
    Build farm BQM with objective $\max \sum_{c \in \text{selected\_crops}} b_c \cdot a_f \cdot Y_{f,c}$
    Add one-crop constraint: $\sum_c Y_{f,c} \leq 1$
    Add U-linking: $Y_{f,c} \leq U_c^*$ encoded as penalty
    $\mathbf{Y}_f^* \leftarrow$ QPU.sample(BQM_f)
    $\mathbf{Y}[f, :] \leftarrow \mathbf{Y}_f^*$
end for
return $\mathbf{x} = (\mathbf{Y}, \mathbf{U}^*)$

- **Global constraint enforcement**: Master ensures food group diversity
- **QPU calls**: $1 + |\mathcal{F}|$ (one master + one per farm)

2.7 Method 7: Spectral Clustering

2.7.1 Description

Spectral clustering uses the eigenvectors of the graph Laplacian to partition variables, grouping tightly connected components while cutting weak connections.

2.7.2 Mathematical Formulation

Given interaction graph $G = (\mathcal{V}, E)$, compute:

Adjacency matrix: $A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$

Degree matrix: $D_{ii} = \sum_j A_{ij}$

Normalized Laplacian: $\mathcal{L} = I - D^{-1/2} A D^{-1/2}$

Spectral embedding: Compute eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ corresponding to smallest eigenvalues

Clustering: Apply k-means on the embedding matrix $V = [\mathbf{v}_1 | \cdots | \mathbf{v}_k]$

2.7.3 Algorithm

2.7.4 Characteristics

- **Spectral properties**: Uses graph spectrum for optimal cuts
- **Balanced partitions**: k-means encourages similar partition sizes
- **Computationally expensive**: Eigenvalue decomposition $O(|\mathcal{V}|^3)$
- **Fixed partition count**: User specifies k

Algorithm 9 Spectral Clustering Decomposition

Require: Interaction graph $G = (\mathcal{V}, E)$, number of clusters k
Ensure: Partition set $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$
1: Construct adjacency matrix A from G
2: Compute degree matrix D
3: Compute normalized Laplacian: $\mathcal{L} = I - D^{-1/2}AD^{-1/2}$
4: Compute k smallest eigenvectors: $V = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$
5: Apply k-means clustering on rows of V to get cluster assignments
6: **for** cluster $i = 1$ to k **do**
7: $\quad \mathcal{P}_i \leftarrow$ variables assigned to cluster i
8: **end for**
9: **return** $\mathcal{P}$

2.8 Method 8: HybridGrid Decomposition

2.8.1 Description

HybridGrid partitioning creates a 2D grid structure by dividing both farms *and* crops simultaneously. This produces many small partitions that are easy to embed while maintaining local constraint coherence. This method emerged as the best-performing pure QPU approach in our benchmarks.

2.8.2 Mathematical Formulation

Given group sizes k_F for farms and k_C for crops, create a grid of partitions:

$$\mathcal{P}_{\text{HybridGrid}} = \{\mathcal{P}_{(i,j)} : i \in [1, \lceil |\mathcal{F}|/k_F \rceil], j \in [1, \lceil |\mathcal{C}|/k_C \rceil]\} \cup \{\mathcal{P}_U\} \tag{19}$$

where each grid cell contains:

$$\mathcal{P}_{(i,j)} = \{Y_{f,c} : f \in \mathcal{F}_{[k_F(i-1)+1:k_F \cdot i]}, c \in \mathcal{C}_{[k_C(j-1)+1:k_C \cdot j]}\} \tag{20}$$

For example, with $k_F = 5$ farms and $k_C = 9$ crops:

- Partition size: $5 \times 9 = 45$ variables (very easy to embed)
- For 100 farms: $20 \times 3 = 60$ grid partitions + 1 U partition
- For 1000 farms: $200 \times 3 = 600$ grid partitions + 1 U partition

2.8.3 Algorithm

2.8.4 Key Advantages

1. **Small partition size**: $k_F \times k_C$ variables (typically 27-65) ensures easy embedding
2. **No embedding failures**: Partitions fit easily on QPU Pegasus topology
3. **Consistent performance**: Predictable partition sizes across all problem scales
4. **Constraint locality**: Each partition covers a coherent subset of the problem
5. **Linear scaling**: Number of partitions scales as $O(|\mathcal{F}|/k_F)$

2.8.5 Empirical Performance

HybridGrid achieves the best quality-to-time ratio among pure QPU methods:

Algorithm 10 HybridGrid Decomposition

Require: Farm group size k_F, crop group size k_C
Ensure: Partition set $\mathcal{P}$

$\mathcal{P} \leftarrow \emptyset$
for $i = 0$ to $\lfloor |\mathcal{F}|/k_F \rfloor$ **do**
  $\mathcal{F}_i \leftarrow \{f_{k_F \cdot i+1}, \ldots, f_{\min(k_F(i+1),|\mathcal{F}|)}\}$
  for $j = 0$ to $\lfloor |\mathcal{C}|/k_C \rfloor$ **do**
    $\mathcal{C}_j \leftarrow \{c_{k_C \cdot j+1}, \ldots, c_{\min(k_C(j+1),|\mathcal{C}|)}\}$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{\{Y_{f,c} : f \in \mathcal{F}_i, c \in \mathcal{C}_j\}\}$
  end for
end for
$\mathcal{P}_U \leftarrow \{U_c : c \in \mathcal{C}\}$
return $\mathcal{P} \cup \{\mathcal{P}_U\}$

Table 1: HybridGrid Performance Across Scales

Config	k_F	k_C	Gap (%)	QPU Time (s)	Violations
HybridGrid(5,9) 100 farms	5	9	9.6	10.8	0
HybridGrid(5,9) 500 farms	5	9	10.0	53.3	0
HybridGrid(5,9) 1000 farms	5	9	**10.0**	105.2	0
HybridGrid(10,9) 100 farms	10	9	16.7	6.0	0
HybridGrid(10,9) 500 farms	10	9	17.9	30.8	1
HybridGrid(10,9) 1000 farms	10	9	18.1	61.5	0
Multilevel(10) 1000 farms	10	–	39.9	26.8	0

2.8.6 Why HybridGrid Works

The key insight is that the one-crop-per-farm constraint is *local* within each partition when partitioning by both farms and crops. Unlike Multilevel which creates large partitions with all 27 crops per farm (leading to 270 variables for 10 farms), HybridGrid creates smaller coherent blocks:

- **Multilevel(10)**: 10 farms × 27 crops = 270 variables per partition
- **HybridGrid(5,9)**: 5 farms × 9 crops = 45 variables per partition

The smaller partition size means:

- Better embedding with shorter chains
- Fewer chain breaks during annealing
- More accurate representation of the objective function
- Lower error rates in the final solution

3 Comparative Analysis

3.1 Theoretical Comparison

3.2 Performance Metrics

For a problem with $|\mathcal{F}| = 25$ farms and $|\mathcal{C}| = 27$ crops:

Table 2: Theoretical Complexity Comparison

Method	Partitions	Partition Size	Embedding Difficulty	Constraint Preservation
Direct QPU	1	$\|\mathcal{V}\|$	Very High	Perfect
PlotBased	$\|\mathcal{F}\| + 1$	$\|\mathcal{C}\|$	Very Low	Excellent
Multilevel(k)	$\lceil \|\mathcal{F}\|/k \rceil + 1$	$k\|\mathcal{C}\|$	Medium	Good
Louvain	$\sim \|\mathcal{F}\|$	Variable	Low-Medium	Good
CQM-First	$\|\mathcal{F}\| + 1$	$\|\mathcal{C}\|$	Very Low	Excellent
Coordinated	$1 + \|\mathcal{F}\|$	$\|\mathcal{C}\|$	Very Low	Perfect
Spectral(k)	k	$\|\mathcal{V}\|/k$	Medium-High	Fair
HybridGrid(k_F,k_C)	$\lceil \|\mathcal{F}\|/k_F \rceil \cdot \lceil \|\mathcal{C}\|/k_C \rceil$	$k_F \cdot k_C$	**Very Low**	**Excellent**

Table 3: Empirical Results (25 farms, 27 crops)

Method	Objective	Gap (%)	Wall Time (s)	Violations
Ground Truth (Gurobi)	0.4018	0.0	0.06	0
PlotBased_QPU	0.3515	12.5	102.0	0
Multilevel(5)_QPU	0.3786	5.8	75.3	16
Louvain_QPU	0.3730	7.2	100.1	2
CQM-First_PlotBased	0.2818	29.9	97.7	1
Coordinated	0.2916	27.4	84.7	1

3.3 Key Observations

1. **Feasibility vs Solution Quality**: Methods with zero violations (PlotBased) may sacrifice solution quality compared to methods that accept some violations.
2. **Partition Size Trade-off**: Smaller partitions (PlotBased, Coordinated) embed easily but require more QPU calls. Larger partitions (Multilevel) reduce calls but increase embedding difficulty and violations.
3. **Constraint Preservation**: CQM-first approaches maintain explicit constraints better than BQM-first decomposition, but penalty tuning remains challenging.
4. **Scalability**: PlotBased and Coordinated scale linearly with farms ($O(|\mathcal{F}|)$ QPU calls). Direct embedding fails beyond ∼500 variables.
5. **QPU Time vs Wall Time**: Wall time dominated by embedding (∼50-70s) rather than QPU access (∼1-6s), indicating embedding is the bottleneck.

4 Implementation Details

4.1 Conflict Resolution Strategy

When partition solutions are merged, farm assignment conflicts are resolved using benefit comparison:

$$\text{benefit}(f, c) = b_c \cdot a_f \tag{21}$$

If farm f is assigned to both crops c_1 and c_2 from different partitions:

$$\text{Keep } c^* = \arg\max_{c \in \{c_1, c_2\}} \text{benefit}(f, c) \tag{22}$$

This greedy strategy maximizes local benefit but may not be globally optimal.

4.2 Lagrange Multiplier Selection

For CQM-to-BQM conversion, Lagrange multipliers must be carefully chosen:

- **One-crop constraint**: $\lambda_{\text{one-crop}} = 10 \times \max_c b_c$ (strong penalty)
- **Food group constraints**: $\lambda_{\text{food-group}} = 50 \times \max_c b_c$ (very strong)
- **U-Y linking**: Encoded structurally in partitioning rather than penalties

4.3 QPU Annealing Parameters

Standard parameters used across all methods:

- **num_reads**: 1000 (number of annealing cycles)
- **annealing_time**: 20 μs (default for Advantage system)
- **chain_strength**: Auto-calculated (typically 2-3× max coupling)

5 Conclusions and Recommendations

5.1 Method Selection Guide

- **Small problems ($|\mathcal{V}| < 300$)**: Use Direct QPU for optimal results
- **Best pure QPU quality**: Use **HybridGrid(5,9)** (achieves 10% gap at 1000 farms)
- **Fast pure QPU**: Use **HybridGrid(10,9)** (faster with 18% gap)
- **Feasibility critical**: Use PlotBased or HybridGrid (zero violations guaranteed)
- **Legacy comparison**: Multilevel(10) provides baseline but has 40% gap
- **Hierarchical structure**: Use Coordinated for clean master-subproblem formulation
- **Very large scale ($|\mathcal{F}| > 500$)**: HybridGrid scales linearly with consistent quality

5.2 Future Research Directions

1. **Adaptive Lagrange Multipliers**: Automatically tune penalties based on constraint violation feedback
2. **Iterative Refinement**: Use partition solutions as warm starts for global refinement
3. **Hybrid Classical-Quantum**: Embed critical variables on QPU, solve remainder classically
4. **Learning-Based Partitioning**: Use machine learning to predict optimal partition structures
5. **Chain Break Mitigation**: Advanced post-processing to repair chain break errors

5.3 Summary

This technical report presented seven decomposition methods for mapping large-scale optimization problems onto D-Wave quantum annealers. We provided complete mathematical formulations, algorithmic descriptions, and empirical performance analysis. The key insight is that *no single method dominates across all metrics*—method selection requires trading off solution quality, constraint satisfaction, QPU time, and embedding complexity based on problem-specific requirements.

A Notation Reference

Symbol	Definition
$\mathcal{F}$	Set of farms (plots)
$\mathcal{C}$	Set of crops (foods)
$\mathcal{G}$	Set of food groups
$\mathcal{V}$	Set of decision variables
$Y_{f,c}$	Binary: crop c assigned to farm f
U_c	Binary: crop c used anywhere
b_c	Benefit coefficient for crop c
a_f	Area of farm f
G_g	Crops in food group g
m_g, M_g	Min/max crops from group g
$\mathcal{P}_i$	Partition i (subset of variables)
λ	Lagrange multiplier
Q	Set of physical qubits
ϕ	Embedding mapping

B Code Availability

The complete implementation is available in the OQI-UC002-DWave repository:

- `@todo/qpu_benchmark.py` - Main benchmark script with all methods
- `Benchmark Scripts/comprehensive_benchmark.py` - Classical comparison baseline
- `src/` - Problem formulation and utility functions