1 Comparative Optimization Framework

Given a time series of variables $w(1), \ldots, w(T)$, we consider optimization problems where the objective and constraints depend on comparisons between three elements: $w(t)$, 0, and $w(t+1)$.

1.1 Problem Classification

We identify three fundamental comparison cases:

Table 1: Comparison Cases

Case	Logical Condition
1	$w(t) \geq 0 \wedge w(t) \geq w(t+1)$
2	$w(t) \geq 0 \wedge w(t) < w(t+1)$
3	$w(t) < 0$

1.2 Mathematical Formulation

For each time period $t = 1, \ldots, T-1$, we introduce binary indicator variables:

$$\begin{aligned} b_1(t) &= I\{w(t) \geq 0\} \\ b_2(t) &= I\{w(t) \geq w(t+1)\} \end{aligned} \tag{1}$$

The complete optimization problem becomes:

$$\begin{aligned} \min_{w,b} \quad & \sum_{t=1}^{T-1} f_t(w(t), w(t+1), b_1(t), b_2(t)) \\ \text{s.t.} \quad & w(t) \geq -M(1 - b_1(t)) \\ & w(t) \leq M b_1(t) \\ & w(t) - w(t+1) \geq -M(1 - b_2(t)) \\ & w(t) - w(t+1) \leq M b_2(t) \\ & \text{Additional problem-specific constraints} \end{aligned} \tag{2}$$

where M is a sufficiently large constant (Big-M method).

1.3 Implementation Algorithm

Algorithm 1 Comparative Optimization Procedure

1: Initialize binary variables $b_1(t), b_2(t)$ for $t = 1, \ldots, T-1$
2: Define Big-M constant M based on variable bounds
3: **for** each time period $t = 1$ to $T-1$ **do**
4: Add constraints:

$$\begin{aligned} w(t) &\geq -M(1 - b_1(t)) \\ w(t) &\leq M b_1(t) \\ w(t) - w(t+1) &\geq -M(1 - b_2(t)) \\ w(t) - w(t+1) &\leq M b_2(t) \end{aligned}$$

5: Add conditional cost terms:

$$f_t = \begin{cases} f_{\text{case1}}(w(t), w(t+1)) & \text{if } b_1(t) = 1 \wedge b_2(t) = 1 \\ f_{\text{case2}}(w(t), w(t+1)) & \text{if } b_1(t) = 1 \wedge b_2(t) = 0 \\ f_{\text{case3}}(w(t), w(t+1)) & \text{if } b_1(t) = 0 \end{cases}$$

6: **end for**
7: Solve the resulting MILP/MIQP problem

1.4 Special Case Handling

For piecewise linear objectives, we can implement:

$$f_t = (1 - b_1(t)) f_{\text{case3}} + b_1(t) \left[b_2(t) f_{\text{case1}} + (1 - b_2(t)) f_{\text{case2}} \right] \tag{3}$$

1.5 Computational Considerations

- The binary variables double the problem size but enable exact representation
- Choose M carefully: $M = \max_t |w_{\max}(t)| + \epsilon$
- Modern solvers (Gurobi, CPLEX) handle these formulations efficiently

1.6 Example: CVXPY Implementation

```
import cvxpy as cp

T = 10  # Time periods
w = cp.Variable(T)
b1 = cp.Variable(T-1, boolean=True)
b2 = cp.Variable(T-1, boolean=True)
```

```
M = 100  # Appropriate Big-M value

constraints = []
for t in range(T-1):
    constraints += [
        w[t] >= -M*(1-b1[t]),
        w[t] <= M*b1[t],
        w[t] - w[t+1] >= -M*(1-b2[t]),
        w[t] - w[t+1] <= M*b2[t]
    ]

# Example objective: Minimize weighted sum of cases
objective = cp.Minimize(
    sum((1-b1[t])*cp.abs(w[t]) +
        b1[t]*(b2[t]*(w[t]-w[t+1]) +
              (1-b2[t])*(w[t+1]-w[t]))
    for t in range(T-1))
)

prob = cp.Problem(objective, constraints)
prob.solve(solver=cp.GUROBI)
```

2 Optimization with Triple Comparison

2.1 Problem Definition

Given a sequence of variables $w(1), \ldots, w(T)$, we need to solve an optimization problem where the objective and constraints depend on the relative ordering of $w(t)$, 0, and $w(t+1)$ for each $t = 1, \ldots, T-1$. There are six possible cases for each time step:

Table 2: Six Comparison Cases

Case	Ordering	Logical Condition
1	$w(t) \geq w(t+1) \geq 0$	$b_1(t) = 1, b_2(t) = 1, b_3(t) = 1$
2	$w(t) \geq 0 \geq w(t+1)$	$b_1(t) = 1, b_2(t) = 0, b_3(t) = 1$
3	$0 \geq w(t) \geq w(t+1)$	$b_1(t) = 0, b_2(t) = 0, b_3(t) = 1$
4	$w(t+1) > w(t) \geq 0$	$b_1(t) = 1, b_2(t) = 1, b_3(t) = 0$
5	$w(t+1) \geq 0 > w(t)$	$b_1(t) = 0, b_2(t) = 1, b_3(t) = 0$
6	$0 > w(t+1) > w(t)$	$b_1(t) = 0, b_2(t) = 0, b_3(t) = 0$

2.2 Mathematical Formulation

$$
\begin{aligned}
\text{Binary variables:} \quad & b_1(t) = I\{w(t) \geq 0\}, \quad b_2(t) = I\{w(t+1) \geq 0\}, \\
& b_3(t) = I\{w(t) \geq w(t+1)\} \\
\text{Case variables:} \quad & c_k(t) \in \{0,1\}, \quad k = 1, \ldots, 6 \\
\text{Constraints:} \quad & \sum_{k=1}^{6} c_k(t) = 1 \quad \forall t \\
& \text{Big-M constraints for } b_1, b_2, b_3 \\
& \text{Linearized case activation constraints} \\
\text{Objective:} \quad & \min \sum_{t=1}^{T-1} \left[\sum_{k=1}^{6} f_k(w(t), w(t+1)) \cdot c_k(t) \right]
\end{aligned}
\tag{4}
$$

2.3 Complete Implementation

Algorithm 2 Triple Comparison Optimization

1: Initialize $w(t)$ for $t = 1, \ldots, T$
2: Initialize binary variables $b_1(t), b_2(t), b_3(t)$ for $t = 1, \ldots, T-1$
3: Initialize case variables $c_k(t)$ for $k = 1, \ldots, 6$ and $t = 1, \ldots, T-1$
4: Set Big-M constant $M \gg \max |w(t)|$
5: **for** $t = 1$ to $T-1$ **do**
6: Add Big-M constraints:

$$
\begin{aligned}
w(t) &\geq -M(1 - b_1(t)), \quad w(t) \leq M b_1(t) \\
w(t+1) &\geq -M(1 - b_2(t)), \quad w(t+1) \leq M b_2(t) \\
w(t) - w(t+1) &\geq -M(1 - b_3(t)), \quad w(t) - w(t+1) \leq M b_3(t)
\end{aligned}
$$

7: Add case activation constraints:

$$
\begin{aligned}
& c_1(t) \leq b_1(t), c_1(t) \leq b_2(t), c_1(t) \leq b_3(t) \\
& c_1(t) \geq b_1(t) + b_2(t) + b_3(t) - 2 \\
& \quad \vdots \quad \text{(similar for cases 2-6)} \\
& c_6(t) \leq 1 - b_1(t), c_6(t) \leq 1 - b_2(t), c_6(t) \leq 1 - b_3(t) \\
& c_6(t) \geq (1 - b_1(t)) + (1 - b_2(t)) + (1 - b_3(t)) - 2
\end{aligned}
$$

8: Add one-hot constraint: $\sum_{k=1}^{6} c_k(t) = 1$
9: **end for**
10: Solve the MILP/MIQP problem

2.4 Python Code Skeleton

```
import cvxpy as cp
```

```
T = 10  # Time periods
w = cp.Variable(T)
b1 = cp.Variable(T-1, boolean=True)
b2 = cp.Variable(T-1, boolean=True)
b3 = cp.Variable(T-1, boolean=True)
c = cp.Variable((6, T-1), boolean=True)
M = 1e5

constraints = []
for t in range(T-1):
    # Big-M constraints
    constraints += [
        w[t] >= -M*(1-b1[t]), w[t] <= M*b1[t],
        w[t+1] >= -M*(1-b2[t]), w[t+1] <= M*b2[t],
        w[t]-w[t+1] >= -M*(1-b3[t]), w[t]-w[t+1] <= M*b3[t]
    ]

    # Case constraints
    constraints += [
        c[0,t] <= b1[t], c[0,t] <= b2[t], c[0,t] <= b3[t],
        c[0,t] >= b1[t]+b2[t]+b3[t]-2,
        # ... cases 2-5 ...
        c[5,t] <= 1-b1[t], c[5,t] <= 1-b2[t], c[5,t] <= 1-b3[t],
        c[5,t] >= (1-b1[t])+(1-b2[t])+(1-b3[t])-2,
        cp.sum(c[:,t]) == 1
    ]

objective = cp.Minimize(cp.sum(w) + cp.sum(c[0,:])*10)  # Example
prob = cp.Problem(objective, constraints)
prob.solve(solver='GUROBI')
```

2.5 Key Features

- **Exact Modeling**: Captures all 6 cases without approximation
- **Computational Efficiency**: Linear in number of time periods
- **Flexibility**: Easy to add case-specific objectives/constraints
- **Solver Compatibility**: Works with Gurobi, CPLEX, etc.

3 Problem Formulation

We solve a multi-period portfolio optimization problem where transaction costs and risk models depend on the relative ordering of weights across time periods.

For each asset i and time period t, we consider six possible cases comparing $w_i(t)$, $w_i(t+1)$, and 0.

3.1 Six Comparison Cases

Table 3: Weight Transition Cases

Case	Condition	Binary Values	Description
1	$w_i(t) \geq w_i(t+1) \geq 0$	$b_1 = 1, b_2 = 1, b_3 = 1$	Decreasing positive position
2	$w_i(t) \geq 0 \geq w_i(t+1)$	$b_1 = 1, b_2 = 0, b_3 = 1$	Moving from long to short
3	$0 \geq w_i(t) \geq w_i(t+1)$	$b_1 = 0, b_2 = 0, b_3 = 1$	Decreasing short position
4	$w_i(t+1) > w_i(t) \geq 0$	$b_1 = 1, b_2 = 1, b_3 = 0$	Increasing long position
5	$w_i(t+1) \geq 0 > w_i(t)$	$b_1 = 0, b_2 = 1, b_3 = 0$	Moving from short to long
6	$0 > w_i(t+1) > w_i(t)$	$b_1 = 0, b_2 = 0, b_3 = 0$	Increasing short position

4 Mathematical Model

4.1 Decision Variables

- $w \in R^{n \times T}$: Portfolio weights
- $b_1, b_2, b_3 \in \{0, 1\}^{n \times (T-1)}$: Binary indicators
- $c \in \{0, 1\}^{6 \times n \times (T-1)}$: Case activation variables

4.2 Constraints

For each asset i and time t:

$$\text{Big-M:} \quad \begin{cases} w_i(t) \geq -M(1 - b_1(i,t)) \\ w_i(t) \leq M b_1(i,t) \\ w_i(t+1) \geq -M(1 - b_2(i,t)) \\ w_i(t+1) \leq M b_2(i,t) \\ w_i(t) - w_i(t+1) \geq -M(1 - b_3(i,t)) \\ w_i(t) - w_i(t+1) \leq M b_3(i,t) \end{cases}$$

$$\text{Case 1:} \quad \begin{cases} c(1,i,t) \leq b_1(i,t) \\ c(1,i,t) \leq b_2(i,t) \\ c(1,i,t) \leq b_3(i,t) \\ c(1,i,t) \geq b_1(i,t) + b_2(i,t) + b_3(i,t) - 2 \end{cases}$$

$$\vdots \quad \text{(Similar for cases 2-6)}$$

$$\text{Portfolio:} \quad \begin{cases} \sum_{i=1}^{n} w_i(t) = 1 \\ -0.1 \leq w_i(t) \leq 0.5 \end{cases}$$

4.3 Objective

Maximize risk-adjusted returns:

$$\max \underbrace{\sum_{t=1}^{T-1} r_t^\top w_{t+1}}_{\text{Returns}} - \underbrace{\sum_{k=1}^{6} \sum_{i,t} c(k,i,t) w_t^\top Q_k w_t}_{\text{Risk}} - \underbrace{0.01 \sum_{t=1}^{T-1} \|w_{t+1} - w_t\|_1}_{\text{Transactions}}$$

5 Implementation

Listing 1: MPO MIQP Implementation

```
import cvxpy as cp
import numpy as np

# Problem parameters
T = 5  # Time periods
n = 3  # Number of assets
M = 1e5  # Big-M constant

# Generate random data
np.random.seed(42)
returns = np.random.randn(n, T-1)*0.1 + 0.02
Q = [np.eye(n)*0.1 for _ in range(6)]  # Case-specific risk
```

```python
initial_weights = np.ones(n)/n  # Equal initial allocation

# Variables
w = cp.Variable((n, T))  # Portfolio weights
b1 = cp.Variable((n, T-1), boolean=True)  # w_i(t) >= 0
b2 = cp.Variable((n, T-1), boolean=True)  # w_i(t+1) >= 0
b3 = cp.Variable((n, T-1), boolean=True)  # w_i(t) >= w_i(t+1)
c = cp.Variable((6, n, T-1), boolean=True)  # Case indicators

# Constraints
constraints = [w[:, 0] == initial_weights]  # Initial condition

for i in range(n):
    for t in range(T-1):
        # Big-M constraints
        constraints += [
            w[i,t] >= -M*(1 - b1[i,t]),
            w[i,t] <= M*b1[i,t],
            w[i,t+1] >= -M*(1 - b2[i,t]),
            w[i,t+1] <= M*b2[i,t],
            w[i,t] - w[i,t+1] >= -M*(1 - b3[i,t]),
            w[i,t] - w[i,t+1] <= M*b3[i,t]
        ]

        # Case activation constraints (shown for Case 1)
        constraints += [
            c[0,i,t] <= b1[i,t],
            c[0,i,t] <= b2[i,t],
            c[0,i,t] <= b3[i,t],
            c[0,i,t] >= b1[i,t] + b2[i,t] + b3[i,t] - 2,
            # ... similar for cases 2-6 ...
        ]

        # One-hot encoding
        constraints += [cp.sum(c[:,i,t]) == 1]

# Portfolio constraints
for t in range(T):
    constraints += [
        cp.sum(w[:,t]) == 1,  # Fully invested
        w[:,t] >= -0.1,       # Limited shorting
        w[:,t] <= 0.5         # Concentration limit
    ]

# Objective components
returns_obj = sum(returns[:,t] @ w[:,t+1] for t in range(T-1))
```

```
risk_obj = sum(cp.quad_form(w[:,t], Q[k])*c[k,i,t]
               for k in range(6) for i in range(n) for t in range(T-1))
transaction_cost = 0.01*sum(cp.abs(w[:,t+1] - w[:,t]) for t in range(T-1))

objective = cp.Maximize(returns_obj - risk_obj - transaction_cost)

# Solve
prob = cp.Problem(objective, constraints)
prob.solve(solver=cp.GUROBI, verbose=True)
```

6 Analysis

6.1 Computational Complexity

The problem scales as:

- Variables: $O(nT)$ continuous + $O(nT)$ binary
- Constraints: $O(nT)$ linear + $O(nT)$ quadratic
- Solver time: Depends on branch-and-cut progress

6.2 Solution Interpretation

- Active cases reveal weight transition patterns
- Case-specific risk models allow regime-dependent risk aversion
- Transaction costs penalize frequent rebalancing

7 Problem Formulation

We consider an optimization problem where the objective and constraints depend on the relative ordering of three values at each time step:

$$\{w(t), 0, w(t+1)\}$$

for $t = 1, \ldots, T-1$.

8 Mathematical Modeling

8.1 Six Possible Orderings

For three distinct elements, there are $3! = 6$ possible strict orderings:

Table 4: Weight Transition Cases

Case	Ordering	Description
1	$w(t) \leq 0 \leq w(t+1)$	Negative to positive transition
2	$w(t) \leq w(t+1) \leq 0$	Decreasing negative values
3	$0 \leq w(t) \leq w(t+1)$	Increasing positive values
4	$0 \leq w(t+1) \leq w(t)$	Decreasing positive values
5	$w(t+1) \leq w(t) \leq 0$	Increasing negative values
6	$w(t+1) \leq 0 \leq w(t)$	Positive to negative transition

8.2 Binary Variable Formulation

For each time step t, we introduce:

- Binary variables $b_k(t) \in \{0, 1\}$ for $k = 1, \ldots, 6$
- One-hot encoding constraint: $\sum_{k=1}^{6} b_k(t) = 1$
- Big-M constraints to enforce orderings when $b_k(t) = 1$

9 Implementation

9.1 Base Implementation

```python
import cvxpy as cp
import numpy as np

T = 10  # Time periods
M = 1e5  # Big-M constant

# Variables
w = cp.Variable(T)
b = cp.Variable((6, T-1), boolean=True)

constraints = []

for t in range(T-1):
    # One active case per timestep
    constraints += [cp.sum(b[:, t]) == 1]

    # Case 1: w(t)    0    w(t+1)
    constraints += [
        w[t] <= 0 + M*(1 - b[0, t]),
        0 <= w[t+1] + M*(1 - b[0, t])
    ]
```

```
    # Case 2: w(t)      w(t+1)      0
    constraints += [
        w[t] <= w[t+1] + M*(1 - b[1, t]),
        w[t+1] <= 0 + M*(1 - b[1, t])
    ]

    # ... (similar for cases 3-6) ...

# Example objective
objective = cp.Minimize(cp.sum_squares(w))
prob = cp.Problem(objective, constraints)
prob.solve(solver=cp.GUROBI)
```

9.2 Extended Example with Multiple Assets

```
n = 3  # Number of assets
w = cp.Variable((n, T))
b = cp.Variable((6, n, T-1), boolean=True)

for i in range(n):
    for t in range(T-1):
        constraints += [
            # Case 1 for asset i
            w[i,t] <= 0 + M*(1 - b[0,i,t]),
            0 <= w[i,t+1] + M*(1 - b[0,i,t]),
            # ... other cases ...
        ]
```

10 Case Study

10.1 Portfolio Rebalancing Problem

Consider a portfolio optimization where transaction costs depend on the direction of trades:

- **Case 1/6**: Crossing zero (higher costs)
- **Case 2/5**: Increasing/decreasing short positions
- **Case 3/4**: Increasing/decreasing long positions

10.2 Objective Function

$$\min \sum_{t=1}^{T-1} \left[\underbrace{\|w_{t+1} - w_t\|_2^2}_{\text{Rebalancing}} + \underbrace{\sum_{k=1}^{6} c_k \cdot b_k(t)}_{\text{Case Penalties}} \right]$$

```python
# Case-specific penalties
penalty_weights = np.array([1.0, 0.2, 0.1, 0.1, 0.2, 1.0])
objective = cp.Minimize(
    cp.sum_squares(w[:,1:] - w[:,:-1]) +
    0.1 * cp.sum(cp.multiply(penalty_weights, b))
```

11 Numerical Considerations

11.1 Big-M Selection

Recommended calculation:

$$M = 2 \times \max(\text{expected } |w|) + \epsilon$$

11.2 Tolerances

Add small tolerances to avoid numerical issues:

```python
constraints += [
    w[t] <= 0 + M*(1-b[0,t]) + 1e-6,
    0 <= w[t+1] + M*(1-b[0,t]) - 1e-6
]
```

12 Solution Analysis

12.1 Validation

Post-solution verification:

```python
for t in range(T-1):
    active_case = np.argmax(b[:,t].value)
    print(f"t={t}: Case {active_case+1}")
    print(f"w(t)={w[t].value:.2f}, w(t+1)={w[t+1].value:.2f}")
```