Performance of High-Frequency Pairs-Trading Algorithm using Linear and Quadratic Programs

By: Peter Hindi

Word Count: 2396

Abstract

Arbitrage is a type of investment strategy that involves profiting from assets that trade at different prices across markets, buying an asset in a cheap market and simultaneously selling the same asset in an expensive market. Pairs-trading is a form of arbitrage that involves profiting from divergence in the performance of securities that are expected to perform similarly, betting that their performance will revert to the mean. This research implements an existing high-frequency pairs-trading algorithm using two approaches: A Quadratic Unconstrained Binary Optimization (QUBO) approach, and a Linear Programming (LP) approach. Overall, this research finds that the LP approach carries a significant increase in efficiency compared to the QUBO approach.

Introduction

We begin by defining some key terms below, which will be imperative for understanding subsequent sections of this project.

Arbitrage

Arbitrage is a trading strategy that aims to profit from asset price inefficiencies through the purchase and sale of financial instruments in different markets. There are several types of arbitrage strategies using different asset types and employing varying levels of directional biases. However, all arbitrage strategies aim to exploit price discrepancies while remaining market neutral, managing price movement risk, or both (Burgess, 2023).

Pairs Trading

Pairs trading is a short-term strategy that is considered a form of statistical arbitrage. It involves finding asset pairs with correlated prices and profiting from divergence in their performance by shorting the high-performing asset, and buying the low-performing asset. The strategy effectively anticipates a mean-reversion in the performance of the pair, and profits from the subsequent convergence of asset prices (Gatev et al., 2006).

High-Frequency Trading

High-frequency trading (HFT) is not a trading strategy, but a means of executing trading strategies that generally utilizes the “latest technological advances in market access, market data access and order routing to maximize the returns of established trading strategies” (Gomber et al., 2011, p. 3). While a generally accepted definition of HFT does not currently exist, characteristics of HFT generally include a very high order volume, rapid order cancellation, proprietary trading, short holding periods, and various other factors (Gomber et al., 2011, p. 17).

There are several research papers that have implemented high-frequency pairs trading (HFPT) strategies, including Bowen et al. (2010), Kim (2011), and Wang et al. (2021). Several research papers have also implemented Quadratic Unconstrained Binary Optimization (QUBO) techniques for financial applications using quantum-inspired hardware (Kalra et al., 2018; Buonaiuto et al., 2023; Leclerc et al., 2023; Zhang et al., 2024). Few papers, however, have used the QUBO framework to implement high-frequency arbitrage strategies (Tatsumura et al., 2023; Tatsumura et al., 2020). Tatsumura et al. (2023) execute an HFPT strategy using a QUBO optimization technique with a simulated bifurcation-based combinatorial optimization accelerator (Tatsumura et al., 2023). Tatsumura et al. (2023)'s algorithm is based on a network graph representation of the asset universe.

While QUBO-based optimization allows for the use of specialized quantum-based hardware, which can solve QUBO problems more efficiently than classical computers in many cases, ILP-based optimization carries multiple advantages against QUBO-based optimization (Lee & Jun, 2025). These advantages include more simple implementation of problems using constraints, greater applicability to modern solvers, and improved accessibility through everyday computer hardware.

There is not currently a robust base of literature comparing the efficiency of QUBO and integer linear programming (ILP)-based HFPT approaches using traditional computer hardware. This project aims to help fill this gap.

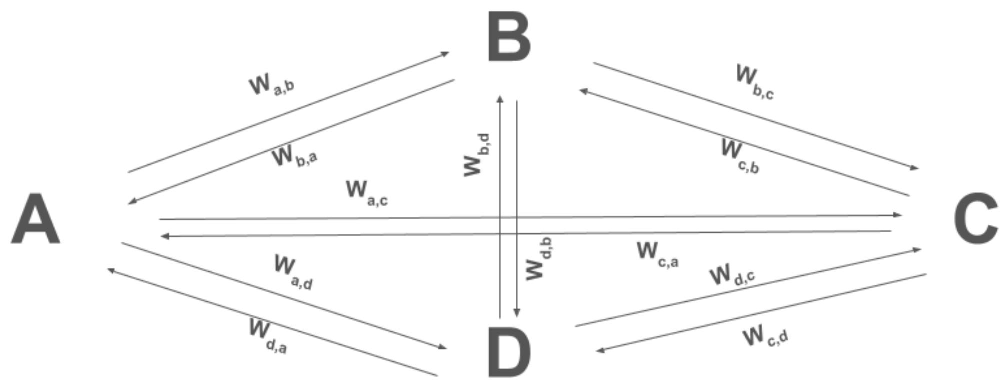
Methodology

In this section, the HFPT algorithm that this research implements using both QUBO and ILP approaches is discussed. The algorithm generally follows the network-based algorithm used in Tatsumura et al. (2023).

Network Graphs

This research utilizes a network graph approach to visualize the asset universe. An example network graph is provided below, representing a universe of four assets. The weight values relate to the interactions between each asset, or node.

Figure 1: Network Graph Representation of the Asset Universe



Cost Function and Optimization

The HFPT algorithm involves the use of a cost function, defined below.

Equation 1: Cost Function

$$H_{cost} = \sum_{i,j} w_{i,j} b_{i,j}$$

The algorithm aims to minimize the cost function, which represents the summed weights of activated edges, by optimizing the decision variable $b_{i,j}$. This optimization is formally defined below,

Equation 2: Cost Optimization

$$Min \sum_{i,j} w_{i,j} b_{i,j}$$

where $w_{i,j}$ denotes the edge weights, and $b_{i,j}$ acts as a binary decision variable for whether an edge is activated.

Weight Calculation

The expanded calculation of the weight variable, $w_{i,j}$, is shown in Equation 3 below.

Equation 3: Weight calculation

$$w_{i,j} = s_{i,j} \times (ask_j - bid_i)$$

Edge weights are the product of a similarity factor, $s_{i,j}$, and the spread between ask_j and bid_i. $s_{i,j}$ is normalized within [0,1] and is calculated on a rolling daily basis. ask_j denotes the lowest ask price for asset j, standardized on the ask price at the beginning of the calendar day. The variable bid_i denotes the highest bid price of asset i, and is also standardized on the bid price at the beginning of the calendar day.

$w_{i,j}$ is at its lowest when $s_{i,j}$ is at its highest and the spread is most negative. A negative spread suggests that asset j has declined in value relative to asset i since the beginning of the day. This indicates that the trader could profit from going long asset j at ask_j and selling asset i at bid_i, profiting when their performance reverts back to the mean.

The similarity factor, $s_{i,j}$, is based on a distance calculation using price data from a rolling one day window. It is based on the reciprocal dynamic time warping (DTW) distance between asset pairs, as shown in Equation 4 below.

Equation 4: Similarity Factor Formula

$$s_{i,j} = 1 \div (DTW_{i,j} + 1)$$

DTW is an algorithm that measures the similarity of time-series data that differ in time frames, where a larger distance denotes lower similarity (Lee, 2019). As such, lower DTW distance leads to a similarity factor closer to one, and a higher DTW distance leads to a similarity factor closer to zero.

Model Output

The intended output of the model is a cycle, where the beginning and end of the cycle is signified by a null node. A given cycle either represents a direct or indirect path, and denotes which assets to be bought and sold. For example, the path (0 -> A -> B -> 0) suggests that the most profitable trade would be to short asset A and go long asset B, where 0 represents the null node. This represents a direct path, because it only includes two assets. An example of a bypass path would be (0 -> A -> B -> C -> 0), which represents shorting A, both going long and shorting B, and going long C. Overall, it represents a pairs-trade between A and C, with the simultaneous purchase and sale of asset B. The bypass path could be more profitable than the direct path, depending on the edge-weights between bypass nodes and surrounding nodes. Bypass paths can include any number of bypass nodes, for example (0 -> A -> B -> C -> D -> 0) denotes a pairs-trade between assets A and D, with bypass nodes B and C.

Below is an example of the model's output in its raw form, as a binary matrix. There are five assets, with the null node noted as "null". The binary values represent $b_{i,j}$ in Equation 1 and Equation 2, namely the activation of edges between each asset. The rows represent i and the columns represent j for each pair.

Figure 2: Matrix Representation of Model Output

	A	B	C	D	E	Null
A	0	0	1	0	0	0
B	0	0	0	0	0	0
C	0	0	0	1	0	0
D	0	0	0	0	0	1
E	0	0	0	0	0	0
Null	1	0	0	0	0	0

The above output denotes the path (0 -> A -> C -> D -> 0), or shorting A, simultaneously buying and shorting C, and buying D.

Rules

To produce cycles that conform to the format described above, we need to set certain rules for the model to follow. Below, we define the rules that must be followed by the model. Note that the tabu list discussed in rule number 4 below represents a limited memory of recent model outputs to forbid duplicate pairs-trade results.

1. The inflow and outflow of each node must be 1 or 0
2. The inflow and outflow of each node must be equal
3. You cannot traverse the same edge twice in different directions
4. You cannot output a duplicate pairs-trade that has been outputted recently (you cannot output a tabu list pair)
5. Each output may only contain one cycle, and cannot contain multiple cycles (you must forbid subtours from occurring)

The implementation of these rules differs depending on the optimization method. In the subsections below, we discuss the implementation of these rules for both the QUBO and ILP optimization methods.

Note that this research uses lazy constraints to enforce the elimination of subtours for both approaches, using Gurobi's callback method to identify and forbid subtours as they arise. This project also adds a constraint to both approaches to ensure that all cycles begin and end with a null node. Thus, this project violates the QUBO's unconstrained nature, while Tatsumura et al. (2023) uses an unspecified external verification technique to ensure these rules are enforced.

Rule Enforcement: QUBO

Below is a penalty term for the QUBO that forbids breaking rules one to four above, introduced by Tatsumura et al. (2023).

Equation 5: Penalty Function

$$H_{penalty} = \sum_{i}\sum_{j\neq j'} b_{i,j}b_{i,j'} + \sum_{j}\sum_{i\neq i'} b_{i,j}b_{i',j} + \sum_{i}(\sum_{j} b_{i,j} - \sum_{j} b_{j,i})^2 + \sum_{i,j} b_{i,j}b_{j,i} + \sum_{i,j} T_{i,j}b_{0,j}b_{i,0}$$

In the above equation, $T_{i,j}$ represents an indicator for whether the pair *i,j* is in the tabu list, and the index *0* represents the null node. The weight between any node and the null node is 0.

Tatsumura et al. (2023) explains the purpose of each term.

> "The first (/second) term forces the outflow (/inflow) of each node to be 1 or less. The third term forces the inflows and outflows of each node to be equal. The fourth term forbids traversing the same edge twice in different directions. The fifth term forbids choosing the pairs in the tabu list $T_{i,j}$. Constraint violations increase the penalty, with $H_{penalty} = 0$ if there are no violations."

Next, we adjust the optimization function in Equation 2 to include this penalty function, which increases the cost function when rules are broken, disincentivizing incorrect outputs. The updated overall cost function for the QUBO is shown below, where m_c and m_p are cost and penalty hyperparameters, respectively. In the results section below, the cost and penalty hyperparameters are set as 1 and 100000000, respectively, to ensure that all of the rules are enforced by the penalty function. The efficiency and quality of results will depend on the balance of hyperparameters between the cost and penalty function.

Equation 6: QUBO Total Cost Function

$$H_{QUBO} = \sum_{i,j,k,l} Q_{i,j,k,l}b_{i,j}b_{k,l} = m_c H_{cost} + m_p H_{penalty}$$

Rule Enforcement: ILP

Because the ILP is capable of using constraints, the implementation of the rules is more simple and involves the simple constraints shown below, where *n* represents the collection of assets in the universe.

Equation 7: Inflow of Each Asset Must be One or Less

$$\sum_{i} b_{i,j} \leq 1, \forall j \in \{1, 2... n\}$$

Equation 8: Outflow of Each Asset Must be One or Less

$$\sum_{j} b_{i,j} \leq 1, \forall i \in \{1, 2... n\}$$

Equation 9: Inflow and Outflow Must be Equal

$$(\sum_{j} b_{i,j} - \sum_{j} b_{j,i}) = 0, \forall i \in \{1, 2... n\}$$

Equation 10: Exclude Pairs in the Tabu List

$$(b_{0,i} + b_{j,0}) \leq 1, \forall i, j \in T_{i,j}$$

Equation 11: Null Node Required

$$\sum_{i} b_{0,i} = 1 \; and \sum_{i} b_{i,0} = 1$$

Results

In this section, we display the results of running the algorithm with both the QUBO and ILP techniques for five assets, showing the time required to run the model for both techniques. Time is denoted in milliseconds. The input data to the model are cryptocurrency futures order-book data, specifically the first five seconds of futures data in May 2023 based on five popular cryptocurrencies: Bitcoin, Binance Coin, Ethereum, XRP, and Solana. The data was taken from Binance's open-access database ("Orderbook Cryptocurrency," 2023).

Figure 3: Model Run Time Table

ILP Approach		
Run #	**Assets**	**Run Time (ms)**
1	5	104.744
2	5	46.855
3	5	119.144
4	5	101.667
5	5	19.156
Average Time (ms)		78.3132

QUBO Approach		
Run #	**Assets**	**Run Time (ms)**
1	5	248.74
2	5	413.442
3	5	292.332
4	5	256.738
5	5	251.585
Average Time (ms)		292.5674

As shown in Figure 3 above, the ILP approach derived significant efficiency gains compared to the QUBO approach with five assets. The average run time across five trials for the QUBO approach is around 293 milliseconds, while the average run time for the ILP is around 78 milliseconds. This makes the ILP almost 275% faster than the QUBO. However, the total objective function for the QUBO was, on average, 1.5% lower than the ILP, suggesting the QUBO produced better results. This is dependent on the hyperparameters used for the QUBO.

For additional information on the data used in the run table, please see Appendix A. For additional information on the hardware and software used for the results, please see Appendix B.

Conclusion

This research finds that the ILP approach performs more efficiently than the QUBO approach when solving Tatsumura et al. (2023)'s HFPT algorithm using traditional computer hardware. While this may highlight an advantage of using the ILP against the QUBO in cases involving traditional computers, it is imperative to understand that results will differ depending on the hardware available. For example, Tatsumura et al. (2023)'s specialized quantum computing hardware achieved a run time of 33 microseconds across 15 assets with the QUBO method, which is orders of magnitude more efficient than the results achieved in this project. The QUBO provided superior results, however, with a 1.5% lower objective function than the ILP approach, on average. This result will vary dependending on the cost and penalty hyperparameters applied to the model.

There are several questions outside the scope of this research that future researchers can investigate. Potential areas of future research include a live trading implementation of the ILP approach and an evaluation of its investment performance compared to the QUBO, the application of ILP and QUBO techniques to other investment strategies, and an analysis of the tradeoffs between the ILP and QUBO techniques in various scenarios, including different numbers of assets, asset classes, and market environments.

References

Bowen, D., Hutchinson, M. C., & O'Sullivan, N. (2010). High-Frequency equity pairs trading: Transaction costs, speed of execution, and patterns in returns. *The Journal of Trading*, *5*(3), 31-38. https://doi.org/10.3905/jot.2010.5.3.031

Buonaiuto, G., Gargiulo, F., De Pietro, G., Esposito, M., & Pota, M. (2023). Best practices for portfolio optimization by quantum computing, experimented on real quantum devices. *Scientific Reports*, *13*(1). https://doi.org/10.1038/s41598-023-45392-w

Burgess, N. (2023). An introduction to arbitrage trading strategies. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4420232

Gatev, E., Goetzmann, W. N., & Rouwenhorst, K. G. (2006). Pairs trading: Performance of a relative value arbitrage rule. *Yale ICF Working Paper No. 08-03*. https://doi.org/10.2139/ssrn.141615

Gomber, P., Arndt, B., Lutat, M., & Uhle, T. E. (2011). High-Frequency trading. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1858626

Kalra, A., Qureshi, F. I., & Tisi, M. (2018). Portfolio asset identification using graph algorithms on a quantum annealer. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3333537

Kim, K. (2011). Performance analysis of pairs trading strategy utilizing high frequency data with an application to KOSPI 100 equities. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1913707

Leclerc, L., Ortiz-Gutiérrez, L., Grijalva, S., Albrecht, B., Cline, J. R. K., Elfving, V. E., Signoles, A., Henriet, L., Del Bimbo, G., Sheikh, U. A., Shah, M., Andrea, L., Ishtiaq, F., Duarte, A., Mugel, S., Cáceres, I., Kurek, M., Orús, R., Seddik, A., . . . M'tamon, D.

(2023). Financial risk management on a neutral atom quantum processor. *Physical Review Research*, *5*(4). https://doi.org/10.1103/physrevresearch.5.043117

Lee, H., & Jun, K. (2025). Range dependent hamiltonian algorithms for numerical QUBO formulation. *Scientific Reports*, *15*(1). https://doi.org/10.1038/s41598-025-93552-x

Lee, H.-S. (2019). Application of dynamic time warping algorithm for pattern similarity of gait. *Journal of Exercise Rehabilitation*, *15*(4), 526-530. https://doi.org/10.12965/jer.1938384.192

Orderbook cryptocurrency futures data. (2023, May). *Binance Market Data*. https://data.binance.vision/?prefix=data/futures/um/daily/bookTicker/

Tatsumura, K., Hidaka, R., Yamasaki, M., Sakai, Y., & Goto, H. (2020). *A currency arbitrage machine based on the simulated bifurcation algorithm for ultrafast detection of optimal opportunity*. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1–5). IEEE. https://doi.org/10.1109/ISCAS45731.2020.9181114

Tatsumura, K., Hidaka, R., Nakayama, J., Kashimata, T., & Yamasaki, M. (2023). Pairs-Trading system using quantum-inspired combinatorial optimization accelerator for optimal path search in market graphs. *IEEE Access*, *11*, 104406-104416. https://doi.org/10.1109/access.2023.3316727

Wang, Q., Teng, B., Hao, Q., & Shi, Y. (2021). High-frequency statistical arbitrage strategy based on stationarized order flow imbalance. *Procedia Computer Science*, *187*, 518-523. https://doi.org/10.1016/j.procs.2021.04.093

Zhang, B., Wang, W., Cheng, F., & Lin, X. (2024). Credit scoring card combination optimization model based on QUBO. *MLPRAE '24: Proceedings of the International Conference on*

Machine Learning, Pattern Recognition and Automation Engineering.

https://doi.org/10.1145/3696687.3696715

Appendix A: Link to Binance Data Used
Link: https://data.binance.vision/?prefix=data/futures/um/monthly/bookTicker/
Folders:

- BTC
  - https://data.binance.vision/data/futures/um/monthly/bookTicker/BTCUSDT/BTCUSDT-bookTicker-2023-05.zip
- BNB
  - https://data.binance.vision/data/futures/um/monthly/bookTicker/BNBUSDT/BNBUSDT-bookTicker-2023-05.zip
- ETH
  - https://data.binance.vision/data/futures/um/monthly/bookTicker/ETHUSDT/ETHUSDT-bookTicker-2023-05.zip
- XRP
  - https://data.binance.vision/data/futures/um/monthly/bookTicker/XRPUSDT/XRPUSDT-bookTicker-2023-05.zip
- SOL
  - https://data.binance.vision/data/futures/um/monthly/bookTicker/SOLUSDT/SOLUSDT-bookTicker-2023-05.zip

Appendix B: Hardware and Software Specifications
Processor: AMD Ryzen 9 5900X 12-Core Processor, 3.70 GHz
RAM: 32.0 GB
System Type: 64-bit operating system, x64-based processor
Operating System: Windows 11 Pro

Optimizer: Gurobi Optimizer version 12.0.1 build v12.0.1rc0
Coding Language: Julia v1.11

Appendix C: GitHub Link for Code
Link: https://github.com/peterhindi/RAWork

Appendix D: Data Reading and Main

```
using Pkg, CSV, DataFrames, DynamicAxisWarping, Distances, Plots, NBInclude

#to pass first transaction_time available from each dataset
initial_time_df = []

#Read in asset-level prices
```

```
btcdf = CSV.read("..\\..\\Crypto Tick Data\\BTCUSDT\\BTCUSDT-bookTicker-2023-05.csv",
DataFrame)
bnbdf = CSV.read("..\\..\\Crypto Tick Data\\BNBUSDT\\BNBUSDT-bookTicker-2023-05.csv",
DataFrame)
ethdf = CSV.read("..\\..\\Crypto Tick Data\\ETHUSDT\\ETHUSDT-bookTicker-2023-05.csv",
DataFrame)
xrpdf = CSV.read("..\\..\\Crypto Tick Data\\XRPUSDT\\XRPUSDT-bookTicker-2023-05.csv",
DataFrame)
soldf = CSV.read("..\\..\\Crypto Tick Data\\SOLUSDT\\SOLUSDT-bookTicker-2023-05.csv",
DataFrame)

twoddf = [[btcdf] [bnbdf] [ethdf] [xrpdf] [soldf]]

for (loop_index, df) in enumerate(twoddf)
    df =
(select(df,[:"best_bid_price",:"best_ask_price",:"best_ask_qty",:"best_bid_qty",:"event_time",
"transaction_time"]))
    push!(initial_time_df, first(df[!,"transaction_time"]))
    twoddf[loop_index] = df

end

#constant to subtract from transaction_time column for each dataframe
subtract_time = minimum(initial_time_df)

loop_index = 1

for (loop_index, df) in enumerate(twoddf)

    df[!, :transaction_time] = df[!, :transaction_time] .- subtract_time
    sort!(df, :transaction_time)
    df = filter(row -> row.transaction_time < 172800000, df)
    df[!, "total_quantity"] = df[!,"best_ask_qty"]+ df[!,"best_bid_qty"]
    df[!, "weighted_avg_price"] = ((df[!,"best_ask_qty"].*df[!,"best_ask_price"]) +
(df[!,"best_bid_qty"].*df[!,"best_bid_price"]))./df[!, "total_quantity"]

    #convert milliseconds to days
    transform!(df, :transaction_time => ByRow(x -> floor(Int, x / 86400000)) => :day_group)
    #group dataframe by day
    gdf = groupby(df, :day_group)
```

```
    #within each group, divide the price by the first price (being done daily now)
    transform!(gdf, :weighted_avg_price => (prices -> prices ./ first(prices)) => :price_index)

    twoddf[loop_index] = df

end

@nbinclude("TSP Pairs Trade Parameterized.ipynb")
@nbinclude("Similarity Factor & Bid-Ask Prices Parameterized.ipynb")
include("QUBO Pairs Trade Parameterized.jl")


level_2_df = []

for df in twoddf
    push!(level_2_df, filter(row -> row.transaction_time < 5000, df))
end

similarity = similarityfactor(level_2_df)

bid_price_df = []
ask_price_df = []

for df2 in level_2_df
    push!(bid_price_df, last(df2[!, "best_bid_price"]))
    push!(ask_price_df, last(df2[!, "best_ask_price"]))
end

@time begin
QUBO_Pairs_Trade(similarity, ask_price_df, bid_price_df, 6)
println("")
println("above is the QUBO result")
end

@time begin
display(TSP_Pairs_Trade(similarity, ask_price_df, bid_price_df, 15))
println("")
println("above is the ILP result")
end
```

Appendix E: Similarity Factor Calculation Code

```
#Import packages
using JuMP, Pkg, CSV, DataFrames, Statistics, Plots, Ipopt, Combinatorics, Distances,
LinearAlgebra, AmplNLWriter, NBInclude, DynamicAxisWarping


#Create matrix of dynamic-time-warping distances between variables for weight calculation and
return it
function similarityfactor(twoddf)
   num_stocks = length(twoddf)
   similarity = zeros(num_stocks, num_stocks)
   for ii in 1:num_stocks
      for j in (ii+1):num_stocks
         a1 = Array(select(twoddf[ii], "price_index"))
         a2 = Array(select(twoddf[j], "price_index"))

         b1 = Array(select(twoddf[ii], "transaction_time"))
         b2 = Array(select(twoddf[j], "transaction_time"))

         #create 2d vector with index price and transaction time
         vector1 = hcat(a1, b1)'
         vector2 = hcat(a2, b2)'
         #calculate dtw
         cost, discard, discard1 = dtw(vector1, vector2)

         similarity[ii,j]= 1/(1+cost)

      end
   end
   similarity = LinearAlgebra.Symmetric(similarity)
   return similarity
End
```

Appendix F: ILP Pairs Trading Model

```
#Import packages
using Pkg, CSV, DataFrames, Statistics, Plots, Ipopt, Combinatorics, Distances, LinearAlgebra,
AmplNLWriter, NBInclude, Gurobi, JuMP, Graphs, GraphRecipes#, PyCall
```

```
@nbinclude("Similarity Factor & Bid-Ask Prices Parameterized.ipynb")
@nbinclude("Cost Function Parameterized.ipynb")

const env = Gurobi.Env()

#Compute forbidden subtours and return list of edges to forbid. Parameters are a list of
components (in this case, cycles) for each callback solution, and a collection of callback edges.
function forbidden_tours(componentlist, cb_edges,null_index)
   #Initialize empty container to add subtour components
   component_container = []
   for component in componentlist
      #Indicator variable for component that includes null node; if null node is present, do not
forbid. Otherwise, forbid the path.
      includes_null = 0
      #if the length of the component is one (if there is no edge/cycle), do not forbid
      if length(component) <= 1
         continue
      #if the length of a component is two (the cycle includes only two nodes such as 5 -> 3 -> 5),
forbid it.
      elseif length(component) == 2
         push!(component_container, component)
         continue
      else
         #if the length of the component is greater than two and includes 5, this is an appropriate
cycle that includes the null node. Do not forbid it.
         for elmt in component
            if elmt == null_index
                  includes_null = 1
            end
         end
      end
      #if the length of the component is greater than two and does not include the null node, then
forbid it because it represents a subtour. Recall that we are also forbidding all cycles that have a
length of two (2 edges)
      if includes_null != 1
         push!(component_container, component)
      end
   end

   #Initialize empty container to store forbidden edges in order of their component.
```

```
    edge_container = []
    #for forbidden components in the callback solution, compute the relevant edges for each
component and add them to the edge_container.
    for component in component_container
        #for each component, initialize a container to store edges for the component.
        edge_set = []
        #for each element within the component, find the relevant edge by searching for the element
within edge source nodes.
        for elmt in component
            for edge in cb_edges
                if src(edge) == elmt
                        push!(edge_set,(src(edge),dst(edge)))
                end
            end
        end
        #push the edge set for the component to the broader container
        push!(edge_container, edge_set)
    end
    #Return the edge container
    return edge_container
end

#set tabu list length for moving window
tabu_list =
[[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1
],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1]]
tabu_index = 1


function tabu_list_push(soln_matrix,tabu_list,tabu_length)
    global tabu_index
    global tabu_list
    #push!(tabu_list, [col_num[1],row_num[1]])

    #dummy column
    index_max = size(soln_matrix)[1]

    #Find solution pair
    dummy_col = soln_matrix[index_max,:]
    dummy_row = soln_matrix[:,index_max]
```

```
    col_num = findall(dummy_col->dummy_col==1, dummy_col)
    row_num = findall(dummy_row->dummy_row==1, dummy_row)

    #revolving index
    if (tabu_index <= tabu_length)
        tabu_list[tabu_index] = [col_num[1],row_num[1]]
        tabu_index += 1
    else
        tabu_list[1] = [col_num[1],row_num[1]]
        tabu_index = 2
    end
    return tabu_list
end

function TSP_Pairs_Trade(similarity, ask_price_df, bid_price_df,tabu_length)

    global tabu_index
    global tabu_list
    global index_size_callback = size(similarity)[1] + 1

    #Initialize our model:
    pairs_trading_model = Model(() -> Gurobi.Optimizer(env))

    index_max = size(similarity)[1]

    @variable(pairs_trading_model, x[i= 1:(index_max+1), j=1:(index_max+1)], Bin)
    @objective(pairs_trading_model, Min, costfunct(x, similarity, ask_price_df, bid_price_df))

    #inflow, outflow, equality
    @constraint(pairs_trading_model, inflow[i in 1:index_max], sum(x[i,:]) <= 1)
    @constraint(pairs_trading_model, outflow[j in 1:index_max], sum(x[:,j]) <= 1)
    @constraint(pairs_trading_model, equality[z in 1:(index_max+1)], sum(x[:,z]) - sum(x[z,:])
== 0)

    #dummy constraint
    @constraint(pairs_trading_model, dummyin, sum(x[(index_max+1),:]) == 1)
    @constraint(pairs_trading_model, dummyout, sum(x[:,(index_max+1)]) == 1)

    #Constraint that trades must be profitable (cost function < 0)
```

```
    #Idea: Modulo Arithmetic (counter that iterates through each element of the list and loops
back to the start)

    @constraint(pairs_trading_model, tabulist[i in 1:size(tabu_list)[1]],
x[(index_max+1),tabu_list[i][1]] + x[tabu_list[i][2],(index_max+1)] <= 1)

    #Lazy constraint to eliminate subtours and short cycles of length two from the solution when
they arise.
    function subtour_elimination_callback(cb_data)
        status = callback_node_status(cb_data, pairs_trading_model)
        if status != MOI.CALLBACK_NODE_STATUS_INTEGER
            return  # Only run at integer solutions
        end

        #Convert callback solution in matrix form to a directed graph
        cb_graph = Graphs.DiGraph(callback_value.(cb_data, pairs_trading_model[:x]))
        #Assign a list of the graph components (in this case, cycles) to the componentlist variable
        componentlist = Graphs.strongly_connected_components(cb_graph)
        #Store edges of the directed graph in a collection variable cb_edges
        cb_edges = collect(Graphs.edges(cb_graph))
        #call the forbidden_tours function to locate forbidden cycles and return the relevant edges
to forbid for each one
        edge_container = forbidden_tours(componentlist, cb_edges, index_size_callback)
        #If the function returns nothing, then do not initialize any lazy constraint
        if length(edge_container) == 0
            return
        else

            #For each forbidden cycle, build a lazy constraint to forbid the relevant edges by ensuring
the sum of edges is less than the length of the component, effectively breaking the cycle.
            for term in edge_container
                edge_limit = length(term)
                con = @build_constraint(sum(pairs_trading_model[:x][edge[1], edge[2]] for edge in
term) <= edge_limit-1)
                MOI.submit(pairs_trading_model, MOI.LazyConstraint(cb_data), con)
            end
        end
        return
    end
```

```
    set_attribute(
        pairs_trading_model,
        MOI.LazyConstraintCallback(),
        subtour_elimination_callback,
    )

    #Optimize model
    optimize!(pairs_trading_model)

    #Build solution matrix
    soln_matrix = round.(Int, value.(x))

    #Add solution to tabu list
    tabu_list = tabu_list_push(soln_matrix,tabu_list,tabu_length)

    if objective_value(pairs_trading_model) >= 0
        return
    else
        return value.(x)
    end
end



#for each in tabu_list
   #@constraint( x[i,j] + x[z,i] <=1)

#lazy constraints
#save model redundancies
```

Appendix G: QUBO Pairs Trading Model

```
#Import packages
using Pkg, CSV, DataFrames, Statistics, Plots, Ipopt, Combinatorics, Distances, LinearAlgebra,
AmplNLWriter, NBInclude, Gurobi, JuMP, Graphs, GraphRecipes#, PyCall

@nbinclude("Similarity Factor & Bid-Ask Prices Parameterized.ipynb")
@nbinclude("Cost Function Parameterized.ipynb")
@nbinclude("Penalty Function Parameterized.ipynb")
```

```
const env = Gurobi.Env()

#Compute forbidden subtours and return list of edges to forbid. Parameters are a list of
components (in this case, cycles) for each callback solution, and a collection of callback edges.
function forbidden_tours(componentlist, cb_edges,null_index)
    #Initialize empty container to add subtour components
    component_container = []
    for component in componentlist
        #Indicator variable for component that includes null node; if null node is present, do not
forbid. Otherwise, forbid the path.
        includes_null = 0
        #if the length of the component is one (if there is no edge/cycle), do not forbid
        if length(component) <= 1
            continue
        #if the length of a component is two (the cycle includes only two nodes such as 5 -> 3 -> 5),
forbid it.
        elseif length(component) == 2
            push!(component_container, component)
            continue
        else
            #if the length of the component is greater than two and includes 5, this is an appropriate
cycle that includes the null node. Do not forbid it.
            for elmt in component
                if elmt == null_index
                        includes_null = 1
                end
            end
        end
        #if the length of the component is greater than two and does not include the null node, then
forbid it because it represents a subtour. Recall that we are also forbidding all cycles that have a
length of two (2 edges)
        if includes_null != 1
            push!(component_container, component)
        end
    end

    #Initialize empty container to store forbidden edges in order of their component.
    edge_container = []
    #for forbidden components in the callback solution, compute the relevant edges for each
component and add them to the edge_container.
```

```
    for component in component_container
        #for each component, initialize a container to store edges for the component.
        edge_set = []
        #for each element within the component, find the relevant edge by searching for the element
within edge source nodes.
        for elmt in component
            for edge in cb_edges
                if src(edge) == elmt
                        push!(edge_set,(src(edge),dst(edge)))
                end
            end
        end
        #push the edge set for the component to the broader container
        push!(edge_container, edge_set)
    end
    #Return the edge container
    return edge_container
end

#set tabu list length for moving window
tabu_list =
[[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1
],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1],[1,1]]
tabu_index = 1


function tabu_list_push(soln_matrix,tabu_list,tabu_length)
    global tabu_index
    global tabu_list
    #push!(tabu_list, [col_num[1],row_num[1]])

    #dummy column
    index_max = size(soln_matrix)[1]

    #Find solution pair
    dummy_col = soln_matrix[index_max,:]
    dummy_row = soln_matrix[:,index_max]

    col_num = findall(dummy_col->dummy_col==1, dummy_col)
    row_num = findall(dummy_row->dummy_row==1, dummy_row)
```

```
    #revolving index
    if (tabu_index <= tabu_length)
        tabu_list[tabu_index] = [col_num[1],row_num[1]]
        tabu_index += 1
    else
        tabu_list[1] = [col_num[1],row_num[1]]
        tabu_index = 2
    end
    return tabu_list
end

function QUBO_Pairs_Trade(similarity, ask_price_df, bid_price_df,tabu_length)

    global tabu_index
    global tabu_list
    global index_size_callback = size(similarity)[1] + 1

    #Initialize our model:
    pairs_trading_model = Model(() -> Gurobi.Optimizer(env))

    index_max = size(similarity)[1]

    #Set hyperparameters
    mc = 1
    mp = 100000000

    #Initialize our model:
    pairs_trading_model = Model(Gurobi.Optimizer)

    @variable(pairs_trading_model, x[i= 1:(index_max+1), j=1:(index_max+1)], Bin)
    @objective(pairs_trading_model, Min, costfunct(x, similarity, ask_price_df, bid_price_df)*mc
+ penaltyfunction(x, tabu_list)*mp)

    #@constraint(pairs_trading_model, tabulist[i in 1:size(tabu_list)[1]],
x[(index_max+1),tabu_list[i][1]] + x[tabu_list[i][2],(index_max+1)] <= 1)

    #Lazy constraint to eliminate subtours and short cycles of length two from the solution when
they arise.
    function subtour_elimination_callback(cb_data)
        status = callback_node_status(cb_data, pairs_trading_model)
```

```
    if status != MOI.CALLBACK_NODE_STATUS_INTEGER
      return  # Only run at integer solutions
    end

    #Convert callback solution in matrix form to a directed graph
    cb_graph = Graphs.DiGraph(callback_value.(cb_data, pairs_trading_model[:x]))
    #Assign a list of the graph components (in this case, cycles) to the componentlist variable
    componentlist = Graphs.strongly_connected_components(cb_graph)
    #Store edges of the directed graph in a collection variable cb_edges
    cb_edges = collect(Graphs.edges(cb_graph))
    #call the forbidden_tours function to locate forbidden cycles and return the relevant edges
to forbid for each one
    edge_container = forbidden_tours(componentlist, cb_edges, index_size_callback)
    #If the function returns nothing, then do not initialize any lazy constraint
    if length(edge_container) == 0
      return
    else

      #For each forbidden cycle, build a lazy constraint to forbid the relevant edges by ensuring
the sum of edges is less than the length of the component, effectively breaking the cycle.
      for term in edge_container
        edge_limit = length(term)
        con = @build_constraint(sum(pairs_trading_model[:x][edge[1], edge[2]] for edge in
term) <= edge_limit-1)
        MOI.submit(pairs_trading_model, MOI.LazyConstraint(cb_data), con)
      end
    end
    return
  end

  set_attribute(
    pairs_trading_model,
    MOI.LazyConstraintCallback(),
    subtour_elimination_callback,
  )

  #Optimize model
  optimize!(pairs_trading_model)

  #Build solution matrix
```

```
    soln_matrix = round.(Int, value.(x))
    #Add solution to tabu list
    tabu_list = tabu_list_push(soln_matrix,tabu_list,tabu_length)
    if objective_value(pairs_trading_model) >= 0
        return
    else
        return value.(x)
    end
end
```