Currency Arbitrage in QUBO Form

The problem of finding the optimal arbitrage opportunity is to find the most profitable cycle in a digraph in which the nodes are the assets and the edges are the conversion rates. This problem can be cast into a quadratic unconstrained binary optimization (QUBO) form, of which the lowest cost solution is the optimal solution that we seek. In the original white paper by 1QBit [1], a QUBO solver of quantum annealing type is considered. We solve the QUBO problems with simulated quantum annealing (SQA) and simulated bifurcation (SB) algorithms implemented on FPGA instead.

Specifically, the cost function is of the following form:

$$C = -\sum_{(i,j)\in E} x_{ij} \log c_{ij} + M_1 \sum_{i\in V} \left(\sum_{j,(i,j)\in E} x_{ij} - \sum_{j,(j,i)\in E} x_{ij} \right)^2 + M_2 \sum_{i\in V} \left(\sum_{j,(i,j)\in E} x_{ij} \right) \left(\sum_{j,(i,j)\in E} x_{ij} - 1 \right)$$

in which the weighted digraph $G = (V, E)$ represents the foreign exchange market, the edge weights c_{ij} corresponds to the exchange rates and the variables $x_{ij} \in \{0, 1\}$ are indicators whether we should take the transaction or not. The parameters M_1 and M_2 are penalty strengths that should be large enough to prevent illegal solutions (e.g., solutions that do not form a cycle). In this way, the lowest cost solution to this problem will be the most profitable simple cycle arbitrage route (or none if no profitable routes exist).

Python Implementation

A python implementation of the above formulation process can be found at `fx_arb_gen.py`. The function `build_Q(problem, cur_lst, M1, M2)` takes in the exchange rates and penalty strengths M_1 and M_2, and returns a matrix Q such that the cost function $C = \mathbf{x}^T Q \mathbf{x}$, where $\mathbf{x} \in \{0, 1\}^N$ denotes the variable vector.

Additionally, the function can also output the cost function in Ising form if `mode='Ising'` is specified. In this case, matrix J and vector $\mathbf{h}$ are returned, so that $C = \mathbf{s}^T J \mathbf{s} + \mathbf{h}^T \mathbf{s}$, $\mathbf{s} \in \{-1, 1\}^N$.

References

[1] https://1qbit.com/whitepaper/arbitrage/

Parameter Optimization Flow

Preparation

- Historical currency exchange rate dataset, preferably with arbitrage opportunity solutions.
- Partition the dataset into two sets: Training set S_T and validation set S_V.

Optimization Flow

1. Define objective function
    a. Separate training set into two parts: problems with/without arbitrage opportunity. Ratio of these two types should stay consistent with our intended use case throughout the following steps if possible.
    b. Randomly choose x problems $\{p_i\}_{i=1}^{x}$ from S_T. A solver gives a solution to each problem p_i, which is compared to the optimal solution that can be achieved (the "real" solution). Each problem gives (solution profitability - optimal profitability) points of score to the objective function. The objective function sums over the scores given by the x problems:

$$obj = \sum_{i=1}^{x}(sol.profit_i - opt.profit_i)$$

$$= \sum_{i=1}^{x} sol.profit_i - \sum_{i=1}^{x} opt.profit_i$$

    c. In case the dataset does not come with the optimal solutions, we can also change our interpretation of the components of the second term $opt.profit_i$ from the *optimal* solution of problem p_i to the best solution *that we've obtained throughout the parameter optimization process*. If at any point during the optimization flow a better solution than the current best is obtained, the best solution should be updated and all previously calculated objective functions be retroactively updated as well.

    d. x can be larger if we have more computational power at hand. 100 may be a good start.

2. Define parameter bounds (guess what is reasonable) (granularities ae included in curly brackets e.g.,{0.1})
    a. $1 < M_1, M_2 < 50$ {1}
    b. $50 < t < 200$ {10} (SB)
    c. $0.001 < dt < 1$ {0.001} (SB)
    d. $0 < c_0 < 1$ {0.01} (SB)
    e. $0.001 < \max|y(t=0)| < 0.1$ {0.001} (random initialization range of y, probably not that important) (SB)
    f. linear $a(t)$ from 0 to 1 (probably doesn't need optimization) (SB)
    g. $4 \leq M \leq 64$ {1} (SQA)
    h. $0 < T < 500$ {1} (SQA)
    i. parametric $\Gamma(t) = At^b$: (SQA)
        i. $0 < A < 100$ {0.1}
        ii. $-4 < b < 0$ {0.001}
3. Use Nelder-Mead or Powell method to conduct bounded optimization on the objective function
    a. Nelder-Mead: https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method
    b. Powell: https://en.wikipedia.org/wiki/Powell%27s_method
    c. An implementation is available in scipy: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html
4. Use the validation set to validate the performance of the new parameters. Compare with previous results.
    a. If any parameter is optimized to near the bounds, consider going back to step 2. and loosen up the bounds.
    b. If the validation results are very similar multiple (e.g., 10) times in a row, terminate the optimization process.

Implementation

To be provided