CS890BR Project

Riley Herman 200352833

August 6, 2020

Abstract

this is the abstract. you know

1 Introduction

In the early days of the stock market, William Stanley Jevons developed a widely used theory on how to predict changes in the market. This is the man that wrote such influential books as The Theory of Political Economy, expounding the marginal utility theory of value that is fundamental to understanding modern economics. A first car has enormous value in comparison to a fifteenth car. Unfortunately for Jevons, his market prediction theory was not so revolutionary. William Peter Hamilton recounts, "[Jevons] propounded the theory of a connection between commercial panics and spots on the sun. ...I said that while Wall Street in its heart believed in a cycle fo panic and prosperity, it did not care if there were enough spots of the sun to make a straight flush" [1]. As much as Jevons' sun spot prediction method sounds absurd, the inherent nature of the stock market is to be unpredictable. His theory is every bit as valid as any other when the theory attempts to explain a complex adaptive system. Michael Mauboussin acknowledges that "[m]ost systems, in nature and in business, are not in equilibrium but rather in constant flux" [2].
Having said that, and despite the orange tabby cat Orlando's success as a stock picker [3], there is little disagreement that making informed portfolio choices is in the best interest of the investor. Therefore, using historical data to generate a set of optimal portfolios is the widely accepted approach. Given the multitude of solving techniques, it is in the investor's best interest to determine the most efficient solver to find the most efficient solution. There are five algorithms presented in this paper: Flower Pollination, Non-dominated Sorting Genetic Algorithm 2 (hereby referred to as NSGA2), Particle Swarm Optimization (PSO), Strength Pareto Evolutionary Algorithm 2 (SPEA2), and traditional Branch and Bound, an exact algorithm the other algorithms are to be compared against [4] [5].

2 Background Knowledge

In order to understand the recommended portfolios and how they are arrived at, one must first understand the problem that is being solved, how it is modelled, what techniques are available and why they were chosen.

2.1 Portfolio Optimization

The problem of portfolio optimization often finds its roots in Harry Markowitz's frequently cited book Portfolio Selection [6]. Though the language may not be the same, what he is proposing is clearly a multi-objective constraint optimization problem. He suggests that portfolios should be weighted using risk and return with respect to a budgetary constraint. Though the problem originates in the field of economics, computing has approached the problem as an excellent candidate for a variety of multi-objective constraint optimization algorithms.
The problem is formulated as follows: given a set of stocks and a budget, one

must use the risk and reward to find an amount of each stock to purchase such that risk is minimal and reward is maximal. Risk and reward are not comparable to each other; the algorithm cannot decide whether a conservative, low reward but also low risk portfolio is better than an aggressive, high risk but high reward portfolio. Thus the solution is a set of options, each one best in its own way. There are several pieces of information that need to be calculated: prices, which are often taken directly from source data; reward, which is NEED TO FILL IN HOW REWARD WORKS; and risk, which was once upon a time calculated as variance but is now often calculated as value-at-risk. Value at risk is a type of downside risk; that is, it attempts to measure risk by estimating the maximum amount lost on the stock [7] [8]. There are a variety of other objectives that could be optimized towards as well: liquiditiy, tax efficiency, etc. However, since the premise of these quantitative optimizations are the same it has been left out of the scope of this project. Another set of objectives that could be optimized towards are qualitative; examples of this may be longevity of the portfolio, where a longer term portfolio may be able to accept more risk than a shorter term portfolio [9], or social responsibility of the companies being traded. Once the problem is formulated in terms of variables (the stocks and how much to buy of each one), constraints (the total amount invested must be below the budget), and objective functions (risk and reward), the problem is ready to be modelled as a constraint optimization problem.

2.2 Constraint Optimization Techniques

A constraint satisfaction problem is defined as a problem with variables, each with a domain and constrained by constraints. The portfolio problem in this form is not particularily useful, because any selection of any amount of the stocks such that the total amount spent does not exceed the budget satifies the constraint. Thus the advice may be to not buy anything at all! A constraint optimization problem is a constraint satisfaction problem with the addition of one or more objective functions. An objective function is a function that is minimized or maximized. For example, Dave is packing his backpack for a trip. He cannot fill it past the point at which he can no longer carry it, but filling it with bricks will not help him on his camping trip. He must fill it with the items that offer the most utility without exceeding the weight constraint. The utility, in this case, must be maximized with respoect given to the constraint.

2.2.1 Branch and Bound

Branch and bound is one example of a solving technique for constraint optimization problems. This is an exact method, and will give the best answer of any algorithm presented here. However, the time and space complexity of this algorithm is of major concern. That is what gave rise to the inexact methods discussed below. Branch and bound can be implemented by the following pseudo-code:

```
define BranchAndBound(node, collection, solutions):
    if node is a leaf node and node is consistent then:
        solutions = solutions + node
        if node->bound > lower_bound then:
            lower_bound := node->bound
        return solutions
    end if
    for variable in node->variables do:
        if variable has value then:
            continue
        end if
        for value in variable->domain do:
            set value for variable
            if node is consistent and lower_bound < node->bound
                then:
                collection = collection + node
    new_node = collection->top
    collection = collection - new_node
    return BranchAndBound(new_node, collection, solutions)
```

A *node* is a representation of the problem. It has several properties: it keeps track of the variables, and each variable keeps track of its domain and current value. It becomes a leaf node when all of its variables have values. When all constraints are satisfied, it is consistent. A node also knows its bound; this is a value that corresponds to its objective functions and allows the algorithm to trim any nodes which would inevitably result in a worse solution than the solutions that have already been obtained. To extend on the example of Dave's camping trip, if Dave has a few items in his backpack that are less than ideal, Dave may realize that even if he adds the most utilitous items left, he won't acheive as good of a backpack as he has already come up with. Therefore there's no point in continuing to add items to this backpack. The collection object can be done in a variety of ways, similar to most searching algorithms. If the collection is a stack, then the algorithm will perform a depth first style of search. It's worth noting that this isn't a true depth first search as it has the added bounding logic. A queue results in a breadth first style fo search, and a priority queue (using a heuritic value) will result in a best first style of search.

2.2.2 Genetic Algorithms

Genetic algorithms are a subset of evolutionary algorithms that are unified through the use of some general nature-inspired functions. These functions include mutation, where a gene changes itself; crossover, where two genes swap part of their sequence; and selection, where the best genes are selected for the next roud of crossover and mutation. Many of these algorithms are explained through the metaphors that they borrow their names from; for example, flower pollination. Flower pollination is, intuitively, based on the pollination of flowering plants. There are four rules that govern this algorithm:

- Biotic, or global pollination
- Abiotic, or local pollination,
- Flower constancy
- Switch probability governing the decision to locally or globally pollinate

Global pollination is similar to the generic crossover genetic function. The candidate that is chosen by the pollinator is chosen using a Lévy flight. A Lévy flight is a type of random walk where the randomly chosen step follows a Lévy distribution, named for mathematician Paul Lévy. It is a left-skewed distribution so the steps will generally not be particularly aggressive. This makes the chances of a valid solution becoming invalid more manageable. Local pollination is similar to the genetic function of mutation. The pollinator steps with generally the same velocity with which it took its previous step, using a uniformly distributed random factor. Flower constancy is a biological phenomenon where a pollinator will often choose the same flower species over and over again, despite other (possibly better) alternatives being present. This is preserved in the algorithm by reproductive ratio, which is proportional to the degree of similar between the pollinator and its pollination candidate. The fourth rule of flower pollination is the switch probability. On any given pollination event, the choice between biotic and abiotic pollination is determined by random chance. The algorithm, using all of these parts, can be expressed by the following pseudocode:

```
define FlowerPollination():
    flowers := initial random population of size n
    g* := find best solutions in flowers
    generation := 0
    while generation < MAX_GENERATIONS do:
        for flower in flowers do:
            if rand > SWITCH_PROBABILITY then:
                l := draw from Levy distribution
                flower->value := flower->value + Gl(g*->value -
                     flower->value)
            else then:
                e := random uniform value in [0, 1]
                flower_a, flower_b := two random flowers from the
                     set of solutions
                flower->value := flower->value + e(flower_a->value
                     - flower_b->value)
            end if
            g* := find best solutions in flowers
        end for
    end while
```

[10] [11]

3 Implementation

3.1 Data

I got the data from Finnhub

3.2 Code

I wrote it in python 3

4 Conclusion

This is the conclusion where we recap all the fun we had this past week.

References

[1] William Peter Hamilton. *The Stock Market Barometer.* New York: Harper and Brothers Publisher, 1921.

[2] Michael J. Mauboussin. “Revisiting Market Efficiency: The Stock Market as a Complex Adaptive System”. In: *Journal of Applied Corporate Finance* 14.4 (2002), pp. 8–16.

[3] Mark King. *Orlando is the cat’s whiskers of stock picking.* `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm`. Accessed: 2020-08-12.

[4] Xin-She Yang. *Nature-Inspired Optimization Algorithms.* London: Elsevier Inc., 2014.

[5] Massimiliano Kaucic, Mojtaba Moradi, and Mohmmad Mirzazadeh. “Portfolio optimization by improved NSGA-II and SPEA 2 based on different risk measures”. In: *Financial Innovation* 1 (2019), pp. 1–28.

[6] Harry Markowitz. “Portfolio Selection”. In: *American Finance Association* 1 (1952), pp. 77–91.

[7] L. Jeff Hong, Zhaolin Hu, and Liwei Zhang. “Conditional Value-at-Risk Approximation to Value-at-Risk Constrained Programs: A Remedy via Monte Carlo”. In: *INFORMS Journal on Computing* 2 (2014), pp. 385–400.

[8] Eva Alfaro Cid. “Several risk measures in portfolio selection: Is it worthwhile?” In: *Revista Espanola de Financiacion y Contabilidad* (2010). DOI: `10.1080/02102412.2010.10779687`.

[9] Jian Xiong et al. “Optimizing Long-Term Bank Financial Products Portfolio Problems with a Multiobjective Evolutionary Approach”. In: *Hindawi* (2020). DOI: `10.1155/2020/3106097`.

[10] Emad Nabil. “A Modified Flower Pollination Algorithm for Global Optimization”. In: *Expert Systems With Applications* 57 (2016), pp. 192–203.

[11] Sankalap Arora and Priyanka Anand. “Chaos-enhanced flower pollination algorithms for global optimization”. In: *Journal of Intelligent and Fuzzy Systems* 33 (2017), pp. 3853–3869.