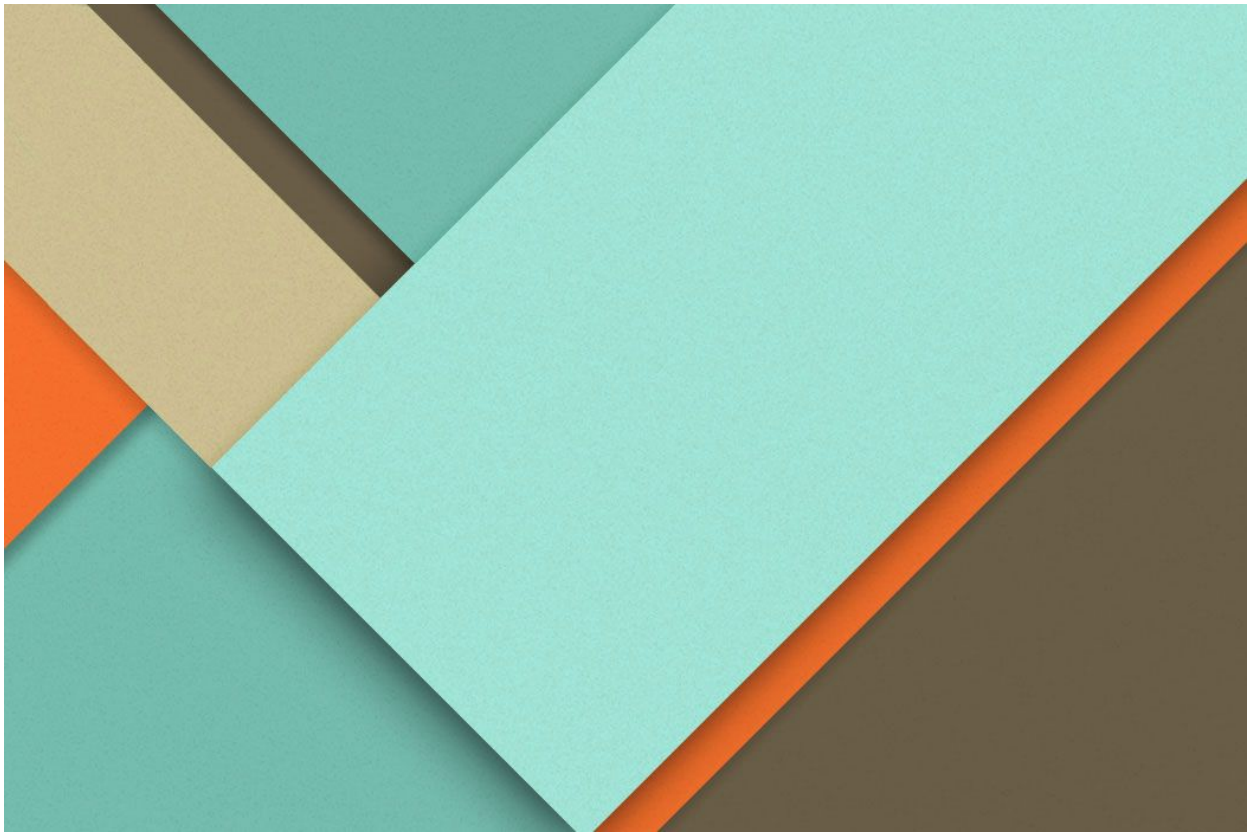


Stock Portfolio CSP

CSP Project

08.14.2017

Abdullah Ali, ID: aliabd11

Roles: (constraint encoding, simulating data (research.py), model encoding, reading input files)

Ammar Akthar, ID: akhtar31

Roles: (constraint encoding, enhancing csp search engine (csp_base.py), acquiring stocks data, adding interactivity)

Project Motivation/Background

The motivation stemmed from whether we can generate portfolios efficiently by representing it as a constraint satisfaction problem. We've considered several different constraints by users (whether they only prefer environmental stocks, where they only want stocks from a certain industry, prefer stocks from a certain region, prefer a minimum price to the stocks, etc.) and see whether we can generate portfolios for a user based on these constraints quickly by representing them as a CSP problem.

CSPs seemed like a good choice for this kind of project as there was a clear state representation, several general purpose algorithms (regular backtracking search, FC and GAC) that we could test to evaluate performance. And as portfolios in the real world would have constraints as per a user's request, the idea of eliminating chunks by identifying value assignments for variables that violate constraints seemed like a smart way to generate portfolios. In addition, unlike some search problems, with CSPs we don't care about the sequence of moves to get to a goal state, simply the feature vector that satisfies the goal making this a great fit for a CSP representation.

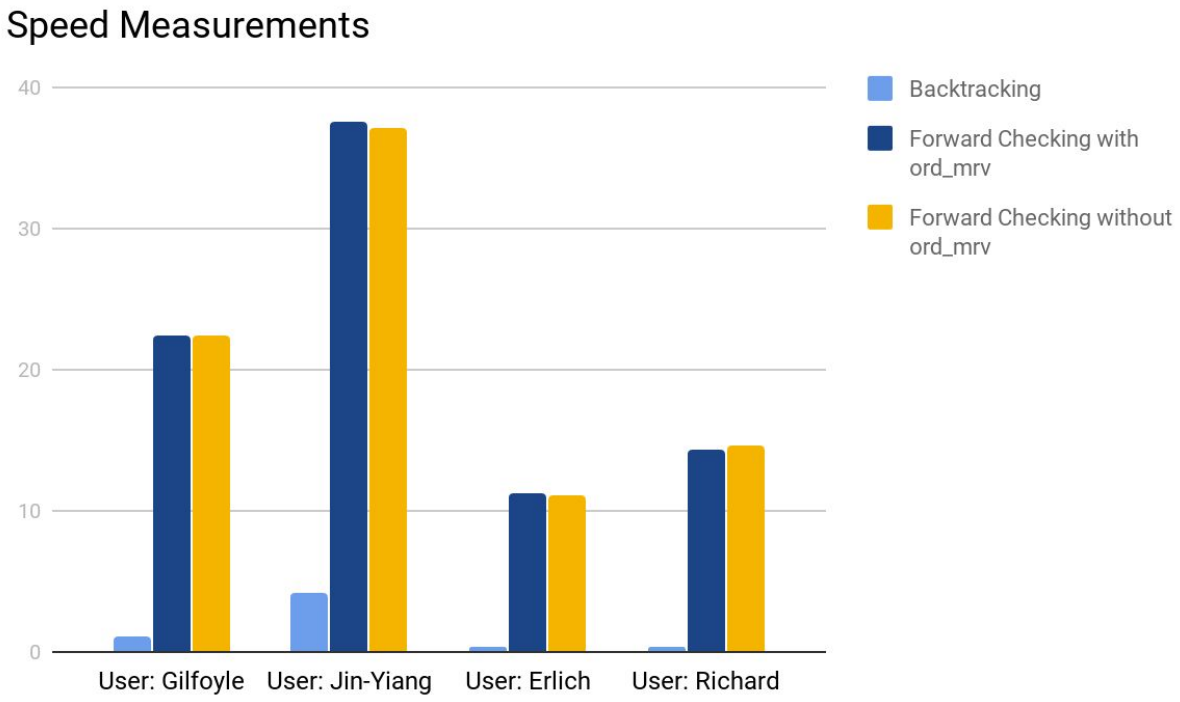
Methods

- We formulated the portfolio as a set of variables [Var1, Var2, Var3...] with the domain of each variable being stock data we've gathered (specifically the symbols (ex. AAL [which refers to American Airlines Group Inc.]). We used both satisfying tuples and formulas to compute the constraints.
- We modified the base CSP code to support formulas in addition to the satisfying tuples representation that was already there.
- Specific constraints that we used included:
    - *ALL-DIFF constraint:* To avoid running into the situation where we get a portfolio of the same stock enumerated multiple times as it satisfies all other constraints.
    - *Green constraint*: Users that only wanted stocks that represented environmentally friendly companies
    - *Industry constraint:* Users that only wanted stocks that came from a certain industry (ex. Technology, Weapons, Food, ...)
    - *Region constraint*: Users that only wanted stocks that came from a certain region of the world (we've limited it to US, Canada and Mexico for practicality purposes)

    - *Spending Limit Constraint:* The total amount of money the user is willing to budget for their portfolio. The price of stocks picked must sum up to a value that is less than the user's specified spending limit. This was done using a formula.
    - *Minimum/Maximum Price Constraint:* Stock prices must conform to price constraints. Min_price_constraint < *Price* < max_price_constraint.
- A pretty loading bar (and feedback to the user while it's computing)! /\/\/
- In addition, multithreading was done to enable some additional speed improvement.

Evaluations and Results

Comparing BT-Search, FC-Search with ord_mrv and FC_search without ord_mrv and four users who we populated the data with. we encountered the following results. The data was tested both with and without the additional multithreading added and relative speed differences were negligible.



We tried out **GAC** but to our surprise, it turned out to be the slowest of the bunch. As we used algorithms that used further propagation, we found that our CSP solved slower than compared to regular backtracking search.

Regular backtracking search was *always* significantly faster and Forward Checking while serviceable was not the optimal choice. The minimum remaining values ordering heuristic did not have a significant effect as can be seen in the graph.

name	**volume_to_buy**	**green**	industry	spending_limit	min_stock_price	max_stock_price	region
Gilfoyle	30	0	Technology	50000	20	500	Canada
Jin-Yiang	50	0	Weapons	40000	25	500	Mexico
Erlich	15	0	Technology	40000	25	600	United States
Richard	20	0	Health	40000	25	500	United States

Above is the user data the measurements are based off. They can be found in the csv file ('user_data') provided.

An example result for the user 'Gilfoyle': Performing BT search

> CSP StocksCSP solved. CPU Time used = 0.9709349999999999
>
> CSP StocksCSP Assignments =
>
> Var--stock_0 = AAL Var--stock_1 = ACOR Var--stock_2 = AFAM Var--stock_3 = AIMT Var--stock_4 = ANCB Var--stock_5 = BKSC Var--stock_6 = BLUE Var--stock_7 = BWLD Var--stock_8 = CARZ Var--stock_9 = CFNL Var--stock_10 = CVCY Var--stock_11 = ECHO Var--stock_12 = EMIF Var--stock_13 = ESGE Var--stock_14 = ESGG Var--stock_15 = FHB Var--stock_16 = FHK Var--stock_17 = FOXF Var--stock_18 = FRME Var--stock_19 = FTC Var--stock_20 = FWP Var--stock_21 = GNRX Var--stock_22 = GTLS Var--stock_23 = HBHCL Var--stock_24 = IBCP Var--stock_25 = INDY Var--stock_26 = INTC Var--stock_27 = IPKW Var--stock_28 = LDRI Var--stock_29 = LFUS

In addition, we attempted multithreading as a way to speed up computation but the effect of that was different depending on the setup. Speed differences generally did not improve by more than a second.

Limitations/Obstacles

1. We ran into problems with representing the constraints. Satisfying tuples weren't feasible for certain constraints such as the spending ones due to the complexity with regards to generating each permutation that satisfies a certain spending constraint.
    a. To resolve this, we resorted to modifying the base code that utilized satisfying tuples and modified it to also use formulas we've specified.

    For example, a constraint saying buy stocks that cost a minimum of $20 could be satisfied by thousands of different stocks costing $20, $20.5, $20.6, $40 etc. We found it physically infeasible to store these less than / greater than numeric constraints in memory, so we modified the constraints checker to use a formula instead looking up satisfying tuples (as there were too many tuples).
2. Gathering data turned out to be very difficult. We managed to get raw stock data with closing prices and other information from Intrinio but we had a difficult time gathering other relevant information about the stocks to perform reasonable constraints on.
    a. To deal with this, we resorted to simulating some of the data (research.py) and adding it into our existing real-world data. In addition, to ensure at least a certain degree of accuracy with regards to the simulated data, when randomly filling in industry or environmental attribute data, the ability to restrict it to a certain percentage of a certain industry (ex. 5% of the stocks represent environmentally friendly companies) was added. Research.py added extra columns like region and industry to each stock ticker, making it possible for us to do much more interesting constraints.

Conclusions:

- Discovering that GAC actually solved our CSP *significantly* slower than Back-Tracking Search and Forward Checking helped illustrate **this** portion of the slide to us.
  - If propagation is slow, this can slow the search down to the point where using propagation makes finding a solution take longer!
  - There is always a tradeoff between searching fewer nodes in the search, and having a higher nodes/second processing rate.
    - We realized the importance of a good state representation and deciding what works best for a given problem.
- We realized the limitations of the satisfying tuples approach and that problem formulation is important when it comes to CSPs.
- In the future, given more time and what we've learned, we would have tried different ordering heuristics to see whether that would have made a difference. In addition, we may have changed the state representation to see whether a different one would have given us even speedier computations.
- Overall, it was a fun project and a neat way to also learn about something we likely would not have learned about otherwise (the financial market!)