Network_Graph

June 30, 2020

1 Building a Minimally Correlated Portfolio with Data Science

The difficulty with modern portfolio theory is that correlations are non-stationary and harbours non-linear effects. On top of that financial data is incredibly sparse and Pearson's correlation only works on time series with equal dimensions. There is accumulating evidence that asset correlation networks, whose nodes are assets and whose edges are the pairwise correlations between the asset's historical returns follows a power-law distribution and show evidence of stationarity. In some sense one can say that these networks are governed by simple scale-free laws. It is known to be hard to forecast financial time series, but maybe the evolution of asset correlation networks is easier to predict if they are driven by simple laws, in which case these laws can be learned by a machine learning algorithm.

In the case of asset correlation networks, we are interested in how volatility spreads between assets between assets and how we can use these insights to optimize our portfolio. To do this we can look at the concept of communicability of complex networks to investigate how things spread and which nodes have the greatest influence in the process. Overall, we will use insights from network science to build centrality-based risk model to generate portfolio asset weights.

1.1 Summary

Using insights from Network Science, we build a centrality-based risk model for generating portfolio asset weights. The model is trained with the daily prices of 31 stocks from 2006-2014 and validated in years 2015, 2016, and 2017. As a benchmark, we compare the model with a portolfio constructed with Modern Portfolio Theory (MPT). Our proposed asset allocation algorithm significantly outperformed both the DIJIA and S&P500 indexes in every validation year with an average annual return rate of 38.7%, a 18.85% annual volatility, a 1.95 Sharpe ratio, a -12.22% maximum drawdown, a return over maximum drawdown of 9.75, and a growth-risk-ratio of 4.32. In comparison, the MPT portfolio had a 9.64% average annual return rate, a 16.4% annual standard deviation, a Sharpe ratio of 0.47, a maximum drawdown of -20.32%, a return over maximum drawdown of 1.5, and a growth-risk-ratio of 0.69.

1.1.1 Asset Diversification and Allocation

The building blocks of a portfolio are assets (resources with economic value expected to increase over time). Each asset belongs to one of seven primary asset classes: cash, equitiy, fixed income, commodities, real-estate, alternative assets, and more recently, digital (such as cryptocurrency and blockchain). Within each class are different asset types. For example: stocks, index funds, and equity mutual funds all belong to the equity class while gold, oil, and corn belong to the commodities class. An emerging consensus in the financial sector is this: a portfolio containing assets of many classes and types hedges against potential losses by increasing the number of revenue streams. In general the more diverse the portfolio the less likely it is to lose money. Take stocks for example. A diversified stock portfolio contains positions in multiple sectors. We call this *asset diversification*, or more simply *diversification*. Below is a table summarizing the asset classes and some of their respective types.

Cash	Equity	Fixed Income	Commodities	Real-Estate	Alternative Assets	Digital
US Dollar	US Stocks	US Bonds	Gold	REIT's	Structured Credit	Cryptocurrencies
Japenese Yen	Foreign Stocks	Foreign Bonds	Oil	Commerical Properties	Liquidations	Security Tokens
Chinese Yaun	Index Funds	Deposits	Wheat	Land	Aviation Assets	Online Stores
UK Pound	Mutual Funds	Debentures	Corn	Industrial Properties	Collectables	Online Media

An investor solves the following (asset allocation) problem: given X dollars and N assets find the best possible way of breaking X into N pieces. By "best possible" we mean maximizing our returns subject to minimizing the risk of our initial investment. In other words, we aim to consistently grow X irrespective of the overall state of the market.

A lower annual standard deviation indicates smaller fluctuations in each revenue stream, and in turn a diminished risk exposure. The "Holy Grail" so to speak, is to (1) find the largest number of assets that are the **least** correlated and (2) allocate X dollars to those assets such that the probability of losing money any given year is minimized. The underlying principle is this: the portfolio most robust against large market fluctuations and economic downturns is a portfolio with assets that are the **most independent** of eachother.

In [1]:
```
#import data manipulation (pandas) and numerical manipulation (numpy) modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
%matplotlib inline


#silence warnings
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Get the data
df = pd.read_csv(r"data/20130102_20200529_daily.csv", index_col=0)
df.head()
```

Out[2]:
```
           Ticker   Open    Low   High  Close       Volume          Name
Day
2013-01-02  ABCB4  14.00  14.00  14.27  14.15    5224632.0    ABC BRASIL
2013-01-02  ALPA4  15.10  14.98  15.30  15.16    2719722.0    ALPARGATAS
2013-01-02  AMAR3  32.55  32.54  33.01  32.63    7420976.0  LOJAS MARISA
2013-01-02  BBAS3  26.00  25.46  26.19  25.80  220234920.0        BRASIL
2013-01-02  BBDC3  34.30  34.30  35.43  35.11   39184241.0      BRADESCO
```

In [3]:
```
stocks = open(r"data/selected_tickers.txt").read().split(",")
np.array(stocks)
```

Out[3]:
```
array(['ABCB4', 'BBAS3', 'BBDC4', 'BRAP4', 'BRML3', 'BRPR3', 'CCRO3',
       'CMIG4', 'CPFE3', 'CSAN3', 'CSNA3', 'CYRE3', 'DTEX3', 'ECOR3',
       'ELET3', 'ENBR3', 'EQTL3', 'EVEN3', 'EZTC3', 'GFSA3', 'GGBR4',
       'GOAU4', 'GOLL4', 'HYPE3', 'IGTA3', 'ITSA4', 'ITUB4', 'LAME4',
       'LIGT3', 'LREN3', 'MRVE3', 'MULT3', 'PETR4', 'RENT3', 'SBSP3',
       'TCSA3', 'TIMP3', 'UGPA3', 'VALE3', 'VIVT4'], dtype='<U5')
```

In [4]:
```
# Filter data with the selected stocks
df = df.loc[df.Ticker.apply(lambda x: x in stocks)]
```

In [5]:
```
df.loc[df.index <= "2017-01-02"].index.max()
```

Out[5]:
```
'2017-01-02'
```

In [6]:
```
# Set the ranges for training and testing
from header import TRAIN_RANGE, TEST_RANGE
print(TRAIN_RANGE)
print(TEST_RANGE)

df_train = df.loc[TRAIN_RANGE[0]:TRAIN_RANGE[1]]
df_train.tail()
```

```
('2013-01-02', '2016-12-29')
('2017-01-02', '2020-05-29')
```

```
Out[6]:            Ticker   Open    Low   High  Close       Volume         Name
        Day
        2016-12-29  TCSA3   2.20   2.13   2.21   2.16    3378789.0      TECNISA
        2016-12-29  TIMP3   7.78   7.63   7.86   7.83   15766673.0  TIM PART S/A
        2016-12-29  UGPA3  67.24  66.60  69.30  68.45   65737657.0     ULTRAPAR
        2016-12-29  VALE3  26.68  25.50  26.85  25.68  123590835.0         VALE
        2016-12-29  VIVT4  43.38  42.99  44.26  44.08   50084766.0  TELEF BRASIL
```

```
In [7]: #testing dataset
        df_validate = df.loc[TEST_RANGE[0]:TEST_RANGE[1]]
        df_validate.tail()
```

```
Out[7]:            Ticker   Open    Low   High  Close        Volume         Name
        Day
        2020-05-29  TCSA3   0.74   0.72   0.75   0.75  6.657817e+06      TECNISA
        2020-05-29  TIMP3  13.46  13.16  13.62  13.62  1.465600e+08  TIM PART S/A
        2020-05-29  UGPA3  17.39  16.83  17.62  17.12  2.456379e+08     ULTRAPAR
        2020-05-29  VALE3  51.40  51.06  53.00  53.00  4.853776e+09         VALE
        2020-05-29  VIVT4  47.39  46.39  47.53  47.14  2.268595e+08  TELEF BRASIL
```

It's always a good idea to check we didn't lose any data after the split.

```
In [8]: #returns True if no data was lost after the split and False otherwise.
        df_train.shape[0] + df_validate.shape[0] == df.shape[0]
```

```
Out[8]: True
```

```
In [9]: # sets each column as a stock and every row as a daily closing price
        df_validate = df_validate.pivot(columns='Ticker', values='Close')
        df_validate.head()
```

```
Out[9]: Ticker      ABCB4  BBAS3  BBDC4  BRAP4  BRML3  BRPR3  CCRO3  CMIG4  CPFE3  \
        Day
        2017-01-02  13.31  27.54  28.80  14.50  12.08   7.47  15.79   7.72  25.26
        2017-01-03  13.88  28.80  30.00  15.11  12.74   7.67  16.39   7.89  25.32
        2017-01-04  14.28  28.65  29.81  14.78  12.72   8.00  16.50   7.67  25.26
        2017-01-05  14.51  28.58  30.14  15.52  13.01   8.05  16.47   7.55  25.21
        2017-01-06  14.65  28.89  30.33  15.25  12.88   7.97  16.23   7.45  25.23

        Ticker      CSAN3  ...  MRVE3  MULT3  PETR4  RENT3  SBSP3  TCSA3  TIMP3  \
        Day                ...
        2017-01-02  37.15  ...  11.05  59.20  14.66  34.94  28.37   2.24   7.73
        2017-01-03  38.88  ...  11.30  61.19  15.50  36.41  28.70   2.35   8.07
        2017-01-04  38.90  ...  11.30  62.12  15.50  36.99  29.53   2.41   8.23
        2017-01-05  38.83  ...  11.35  62.38  15.75  36.43  29.61   2.50   8.33
        2017-01-06  37.93  ...  11.36  61.99  15.66  36.42  29.18   2.46   8.23

        Ticker      UGPA3  VALE3  VIVT4
        Day
```

```
2017-01-02   67.9  25.06  44.08
2017-01-03   69.0  26.17  44.53
2017-01-04   67.9  25.70  44.35
2017-01-05   68.3  26.68  43.60
2017-01-06   68.0  25.97  43.71

[5 rows x 40 columns]
```

In [10]:
```
#creates a DataFrame for each time-series (see In [11])
df_train_close = df_train.pivot(columns='Ticker', values='Close')

#makes a copy of the traning dataset
df_train_close_copy = df_train_close.copy()

df_train_close.head()
```

Out[10]:
```
Ticker       ABCB4  BBAS3  BBDC4  BRAP4  BRML3  BRPR3  CCRO3  CMIG4  CPFE3  \
Day
2013-01-02   14.15  25.80  36.02  34.23  27.75  25.80  19.05  23.00  22.16
2013-01-03   14.19  26.31  38.12  33.87  27.81  25.55  19.39  23.03  21.95
2013-01-04   13.99  26.00  37.45  33.20  27.65  25.79  19.88  21.92  21.29
2013-01-07   14.10  26.15  37.29  32.00  27.41  25.70  19.71  21.19  20.59
2013-01-08   14.25  26.45  37.42  32.00  27.15  25.75  19.70  20.61  20.15

Ticker       CSAN3  ...  MRVE3  MULT3  PETR4  RENT3  SBSP3  TCSA3  TIMP3  \
Day                 ...
2013-01-02   42.50  ...  11.65  59.99  19.69  38.55  86.83   8.22   8.05
2013-01-03   42.24  ...  11.85  59.62  20.40  38.30  85.50   8.28   7.98
2013-01-04   42.11  ...  11.64  59.92  20.48  38.40  85.88   8.17   7.98
2013-01-07   42.30  ...  11.41  59.00  20.08  37.59  86.20   8.10   7.90
2013-01-08   42.75  ...  11.15  59.00  19.50  37.60  85.00   8.00   7.90

Ticker       UGPA3  VALE3  VIVT4
Day
2013-01-02   45.80  44.10  49.62
2013-01-03   45.28  43.35  50.12
2013-01-04   46.70  42.53  50.19
2013-01-07   47.00  41.84  50.20
2013-01-08   47.20  41.51  50.40

[5 rows x 40 columns]
```

In [11]:
```
df_train_close.isnull().sum()
```

Out[11]:
```
Ticker
ABCB4    0
BBAS3    0
BBDC4    0
BRAP4    0
```

```
BRML3    0
BRPR3    0
CCRO3    0
CMIG4    0
CPFE3    0
CSAN3    0
CSNA3    0
CYRE3    0
DTEX3    0
ECOR3    0
ELET3    0
ENBR3    0
EQTL3    0
EVEN3    0
EZTC3    0
GFSA3    0
GGBR4    0
GOAU4    0
GOLL4    0
HYPE3    0
IGTA3    0
ITSA4    0
ITUB4    0
LAME4    0
LIGT3    0
LREN3    0
MRVE3    0
MULT3    0
PETR4    0
RENT3    0
SBSP3    0
TCSA3    0
TIMP3    0
UGPA3    0
VALE3    0
VIVT4    0
dtype: int64
```

In [12]:
```
#idx = df_train_close.loc[df_train_close["RAPT4"].isnull()].index
#df_train_close.loc[idx].
df_train_close.fillna(value=0, inplace=True)
df_train_close_copy.fillna(value=0, inplace=True)
df_validate.fillna(value=0, inplace=True)
```

In [13]:
```
print(df_validate.shape)
print(df_train_close.shape)
```

```
(841, 40)
```

```
(991, 40)
```

In [14]:
```python
# Log-returns
df_train_close.apply(np.log).diff()
```

Out[14]:
```
Ticker         ABCB4     BBAS3     BBDC4     BRAP4     BRML3     BRPR3  \
Day
2013-01-02       NaN       NaN       NaN       NaN       NaN       NaN
2013-01-03  0.002823  0.019575  0.056665 -0.010573  0.002160 -0.009737
2013-01-04 -0.014195 -0.011853 -0.017732 -0.019980 -0.005770  0.009350
2013-01-07  0.007832  0.005753 -0.004282 -0.036814 -0.008718 -0.003496
2013-01-08  0.010582  0.011407  0.003480  0.000000 -0.009531  0.001944
...              ...       ...       ...       ...       ...       ...
2016-12-23  0.020370  0.028064  0.020718  0.000000  0.021883  0.013210
2016-12-26  0.002983  0.015585  0.018143  0.036343  0.010336  0.013038
2016-12-27  0.022092  0.006971 -0.009756 -0.008793  0.015306 -0.022267
2016-12-28  0.017329  0.012355  0.035317  0.030772 -0.008475 -0.011992
2016-12-29 -0.002867  0.014342  0.016340 -0.021979  0.016878  0.005348

Ticker         CCRO3     CMIG4     CPFE3     CSAN3  ...     MRVE3     MULT3  \
Day                                                 ...
2013-01-02       NaN       NaN       NaN       NaN  ...       NaN       NaN
2013-01-03  0.017690  0.001303 -0.009522 -0.006136  ...  0.017022 -0.006187
2013-01-04  0.024957 -0.049398 -0.030530 -0.003082  ... -0.017880  0.005019
2013-01-07 -0.008588 -0.033870 -0.033432  0.004502  ... -0.019957 -0.015473
2013-01-08 -0.000507 -0.027753 -0.021601  0.010582  ... -0.023051  0.000000
...              ...       ...       ...       ...  ...       ...       ...
2016-12-23 -0.002613  0.014736  0.000398 -0.006215  ... -0.000931  0.008906
2016-12-26  0.008466 -0.016086 -0.001990 -0.010628  ...  0.007421 -0.000174
2016-12-27 -0.007813 -0.002706  0.000796 -0.021881  ... -0.000925  0.002431
2016-12-28  0.008461  0.017462 -0.000398  0.055820  ...  0.012868  0.015491
2016-12-29  0.033772  0.026283  0.003975  0.010275  ... -0.000914  0.014076

Ticker         PETR4     RENT3     SBSP3     TCSA3     TIMP3     UGPA3  \
Day
2013-01-02       NaN       NaN       NaN       NaN       NaN       NaN
2013-01-03  0.035424 -0.006506 -0.015436  0.007273 -0.008734 -0.011419
2013-01-04  0.003914  0.002608  0.004435 -0.013374  0.000000  0.030879
2013-01-07 -0.019725 -0.021319  0.003719 -0.008605 -0.010076  0.006403
2013-01-08 -0.029310  0.000266 -0.014019 -0.012423  0.000000  0.004246
...              ...       ...       ...       ...       ...       ...
2016-12-23  0.016284  0.019026  0.002921  0.041385 -0.001313 -0.001213
2016-12-26  0.012561 -0.014603  0.019495  0.004494  0.003934  0.001516
2016-12-27 -0.000694 -0.003832 -0.014767  0.008929 -0.001310  0.005440
2016-12-28  0.025353 -0.022701  0.017980 -0.013423  0.015605  0.013174
2016-12-29  0.006071  0.033277  0.025683 -0.027399  0.010270  0.017835
```

```
Ticker         VALE3     VIVT4
Day
2013-01-02       NaN       NaN
2013-01-03 -0.017153  0.010026
2013-01-04 -0.019097  0.001396
2013-01-07 -0.016357  0.000199
2013-01-08 -0.007918  0.003976
...              ...       ...
2016-12-23 -0.007514 -0.006065
2016-12-26  0.031265  0.005832
2016-12-27 -0.005014  0.000233
2016-12-28  0.031217  0.008107
2016-12-29 -0.038202  0.016699

[991 rows x 40 columns]
```

In [15]:
```
# Remove the seasonality
df_train_close = df_train_close.apply(np.log).diff().iloc[1:].dropna()
df_train_close.head()
```

Out[15]:
```
Ticker         ABCB4     BBAS3     BBDC4     BRAP4     BRML3     BRPR3  \
Day
2013-01-03  0.002823  0.019575  0.056665 -0.010573  0.002160 -0.009737
2013-01-04 -0.014195 -0.011853 -0.017732 -0.019980 -0.005770  0.009350
2013-01-07  0.007832  0.005753 -0.004282 -0.036814 -0.008718 -0.003496
2013-01-08  0.010582  0.011407  0.003480  0.000000 -0.009531  0.001944
2013-01-09 -0.010582  0.009407  0.012746 -0.001564  0.010989  0.023034

Ticker         CCRO3     CMIG4     CPFE3     CSAN3  ...     MRVE3     MULT3  \
Day                                                 ...
2013-01-03  0.017690  0.001303 -0.009522 -0.006136  ...  0.017022 -0.006187
2013-01-04  0.024957 -0.049398 -0.030530 -0.003082  ... -0.017880  0.005019
2013-01-07 -0.008588 -0.033870 -0.033432  0.004502  ... -0.019957 -0.015473
2013-01-08 -0.000507 -0.027753 -0.021601  0.010582  ... -0.023051  0.000000
2013-01-09  0.001522  0.054297  0.022087  0.002103  ...  0.014248  0.005072

Ticker         PETR4     RENT3     SBSP3     TCSA3     TIMP3     UGPA3  \
Day
2013-01-03  0.035424 -0.006506 -0.015436  0.007273 -0.008734 -0.011419
2013-01-04  0.003914  0.002608  0.004435 -0.013374  0.000000  0.030879
2013-01-07 -0.019725 -0.021319  0.003719 -0.008605 -0.010076  0.006403
2013-01-08 -0.029310  0.000266 -0.014019 -0.012423  0.000000  0.004246
2013-01-09  0.009188 -0.014736  0.005748  0.000000 -0.011458 -0.002121

Ticker         VALE3     VIVT4
Day
2013-01-03 -0.017153  0.010026
2013-01-04 -0.019097  0.001396
```

```
2013-01-07 -0.016357  0.000199
2013-01-08 -0.007918  0.003976
2013-01-09  0.004567 -0.007968

[5 rows x 40 columns]
```

In [16]:
```
#imports the dcor module to calculate distance correlation
import dcor

#function to compute the distance correlation (dcor) matrix from a DataFrame and outp
#of dcor values.
from header import df_distance_correlation
```

In [18]:
```
df_train_dcor = df_distance_correlation(df_train_close, stocks)
df_train_dcor.head()
```

Out[18]:
```
          ABCB4     BBAS3     BBDC4     BRAP4     BRML3     BRPR3     CCRO3  \
ABCB4         1  0.437615  0.478087  0.259211  0.335553  0.198967  0.283889
BBAS3  0.437615         1  0.676935  0.366711  0.483295   0.29179  0.419315
BBDC4  0.478087  0.676935         1  0.399386  0.511997  0.331582  0.441633
BRAP4  0.259211  0.366711  0.399386         1  0.318442  0.191926  0.292073
BRML3  0.335553  0.483295  0.511997  0.318442         1  0.429177  0.516971

          CMIG4     CPFE3     CSAN3  ...     MRVE3     MULT3     PETR4  \
ABCB4  0.324016  0.278715  0.297009  ...  0.228829  0.322597  0.354368
BBAS3  0.429405  0.413376  0.393474  ...  0.368762  0.450552  0.546885
BBDC4  0.416678   0.46451   0.44649  ...  0.373357  0.492249  0.583978
BRAP4  0.318885  0.286116   0.28417  ...  0.263662  0.285027  0.469788
BRML3  0.344609  0.440007  0.385689  ...  0.378027  0.623562  0.463181

          RENT3     SBSP3     TCSA3     TIMP3     UGPA3     VALE3     VIVT4
ABCB4  0.280869  0.302807   0.29853    0.2157  0.329163  0.205284   0.28386
BBAS3  0.369564   0.39142  0.381254  0.307464  0.414213  0.318512   0.37403
BBDC4  0.420374   0.42504  0.405225  0.348445  0.463342  0.365086  0.407144
BRAP4  0.285551  0.264666  0.269324  0.326041  0.287027  0.877145  0.309648
BRML3  0.413816   0.36458  0.365397  0.299732  0.425991  0.271807  0.354704

[5 rows x 40 columns]
```

1.1.2 Building a Time-Series Correlation Network with Networkx

In [19]:
```
#imports the NetworkX module
import networkx as nx

# takes in a pre-processed dataframe and returns a time-series correlation
# network with pairwise distance correlation values as the edges
from header import build_corr_nx
```

In [20]:
```
# builds the distance correlation networks for the training data
H_close = build_corr_nx(df_train_dcor, 0.4)
```

In [21]:
```
zero_degree = []
nonzero_degree = []
for t, d in H_close.degree():
    if d == 0:
        zero_degree.append(t)
    else:
        nonzero_degree.append(t)

print(zero_degree)
print(len(zero_degree))
print(nonzero_degree)
print(len(nonzero_degree))
```

```
[]
0
['ABCB4', 'BBAS3', 'BBDC4', 'BRAP4', 'BRML3', 'BRPR3', 'CCRO3', 'CMIG4', 'CPFE3', 'CSAN3', 'CS
40
```

In [22]:
```
# Remove nodes with no connections zero
H_close.remove_nodes_from(zero_degree)
```

In [23]:
```
# Remove from original DF (df_validate will be used for backtesting)
df_train_close.drop(columns=zero_degree, inplace=True)
df_validate.drop(columns=zero_degree, inplace=True)
df_train_close_copy.drop(columns=zero_degree, inplace=True)
```

1.1.3 Plotting a Time-Series Correlation Network with Seaborn

In [24]:
```
# function to display the network from the distance correlation matrix
from header import plt_corr_nx


# function to visualize the degree distribution
from header import hist_plot
```

In [25]:
```
def is_irreducible(H):
    for node, weight in H.degree():
        if weight == 0:
            return False
    return True


def grid_search_threshold(df_dcor, threshold_list):
    for threshold in threshold_list:
        print("Testing for threshold {:,.4f}:".format(threshold))
        H = build_corr_nx(df_dcor, corr_threshold=threshold)
        print("Result: {}".format("Irreducible!" if is_irreducible(H) else "Not irredu
        print()
```

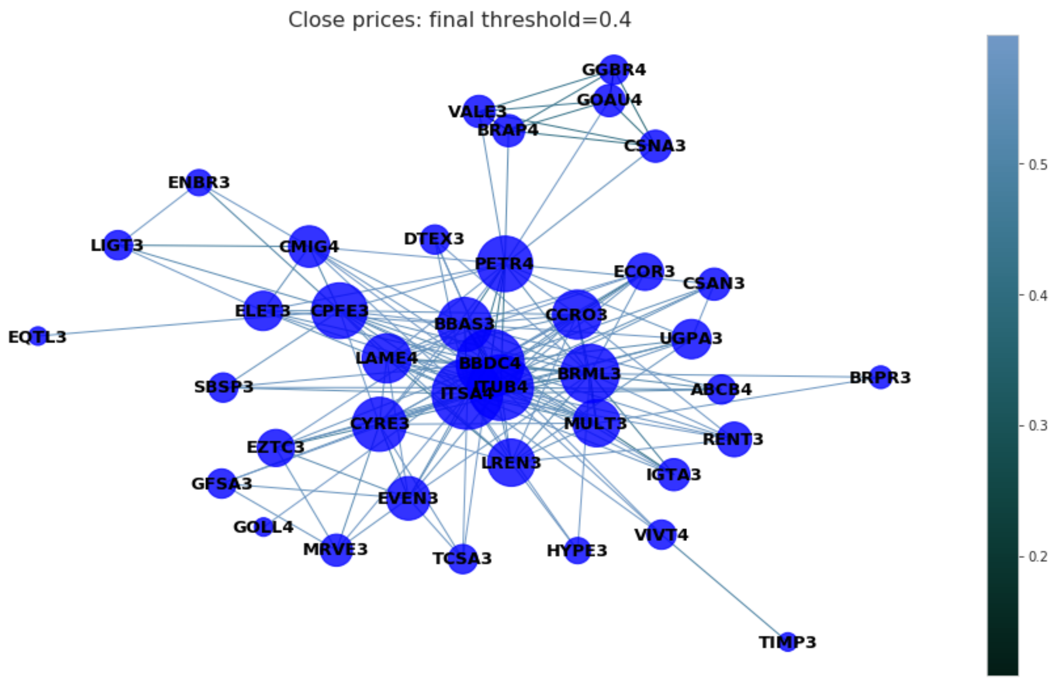
```
threshold_list = [0.0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.325, 0.4, 0.45]
print("Testing for Close price: \n")
grid_search_threshold(df_train_dcor, threshold_list)
```

```
Testing for Close price:

Testing for threshold 0.0000:
Result: Irreducible!

Testing for threshold 0.1000:
Result: Irreducible!

Testing for threshold 0.1500:
Result: Irreducible!

Testing for threshold 0.2000:
Result: Irreducible!

Testing for threshold 0.2500:
Result: Irreducible!

Testing for threshold 0.3000:
Result: Irreducible!

Testing for threshold 0.3250:
Result: Irreducible!

Testing for threshold 0.4000:
Result: Irreducible!

Testing for threshold 0.4500:
Result: Not irreducible!
```

2 Visualizing How A Portfolio is Correlated with Itself (with Physics)

The following visualizations are rendered with the Kamada-Kawai method, which treats each vertex of the graph as a mass and each edge as a spring. The graph is drawn by finding the list of vertex positions that minimize the total energy of the ball-spring system. The method treats the spring lengths as the weights of the graph, which is given by `1 - cor_matrix` where `cor_matrix` is the distance correlation matrix. Nodes seperated by large distances reflect smaller correlations between their time series data, while nodes seperated by small distances reflect larger correlations. The minimum energy configuration consists of vertices with few connections experiencing a repulsive force and vertices with many connections feeling an attractive force. As such, nodes with a larger degree (more correlations) fall towards to the center of the visualization where nodes with a smaller degree (fewer correlations) are pushed outwards. For an overview of physics-based graph visualizations see the Force-directed graph drawing wiki.

```
In [26]: # plots the distance correlation network of the daily opening prices from 2006-2014
         plt_corr_nx(H_close, title='Close prices: final threshold=0.4')
```

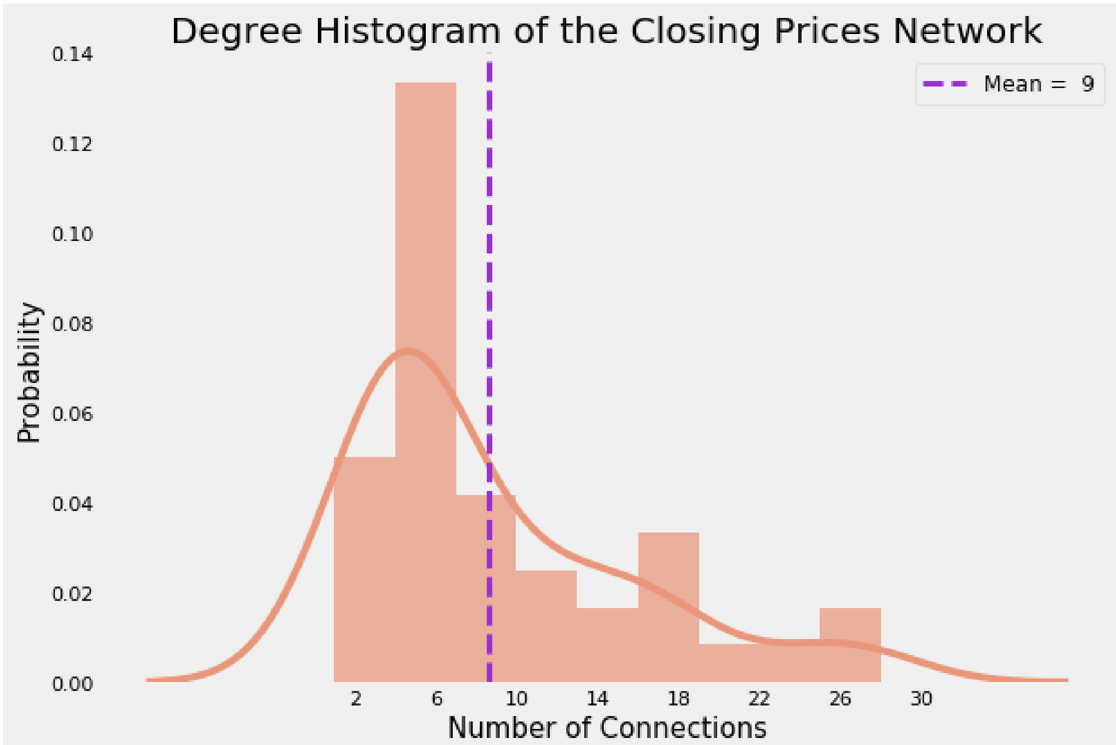


In the above visualization, the sizes of the vertices are proportional to the number of connections they have. The colorbar to the right indicates the degree of disimilarity (the distance) between the stocks. The larger the value (the lighter the color) the less similar the stocks are. Keeping this in mind, several stocks jump out. **JBS**, **CSMG**, **HBOR3**, and all the ones that lie on the periphery of the network with the fewest number of correlations above $\rho_c = 0.325$. On the other hand **BBAS3**, **ITUB4**, **BBDC4**, and **LAME4** sit in the core of the network with the greatest number connections above $\rho_c = 0.325$. It is clear from the closing prices network that our asset allocation algorithm needs to reward vertices on the periphery and punish those nearing the center. In the next code block we build a function to visualize how the edges of the distance correlation network are distributed.

2.1 Degree Histogram

```
In [27]: # plots the degree histogram of the closing prices network
         hist_plot(
             H_close,
             'Degree Histogram of the Closing Prices Network',
             bins=9,
```

```
        xticks=range(2, 31, 4)
    )
```



3 Communicability as a Measure of Relative Risk

We are now in a position to devise a method to compute the allocation weights of our portfolio. To recall, this is the problem:

Given the N assets in our portfolio, find a way of computing the allocation weights w_i, $\left(\sum_{i=1}^{N} w_i = 1\right)$ such that assets more correlated with each other obtain lower weights while those less correlated with each other obtain higher weights.

Theres an infinite number of possible solutions to the above problem. The asset correlation network we built contains information on how our portfolio is interrelated (whose connected to who), but it does not tell us how each asset *impacts* the other or how those impacts travel throughout the network. If, for example, Apple's stock lost 40% of its value wiping out, say, two years of gains, how would this impact the remaining assets in our portfolio? How easily does this kind of behaviour spread and how can we keep our capital isolated from it? We thus seek a measure of "relative risk" that quantifies not only the correlations between assets, but how those correlations mediate perturbations in the portfolio. Our aim, therefore, is twofold: allocate capital inversely proportional to (1) the correlations between assets and (2) proportional to the "impact resistence" of each asset. As luck would have it, there is a centrality measure that does just this! Let us define the relative risk as follows:

Relative Risk of Asset $r = \dfrac{\omega_r}{\sum_{r'=1}^{N} \omega_{r'}}$,

where
$\omega_r = \frac{1}{C}\sum_p \sum_q \frac{G_{prq}}{G_{pq}}$
is the **Communicability Betweenness centrality** (Estrada, *et al.* (2009)) of node r. Here
$G_{prq} = \big(\exp \mathbf{A}\big)_{pq} - \big(\exp\left(\mathbf{A} - \mathbf{E}(r)\right)\big)_{pq}$
is the number of weighted walks involving only node r,
$G_{pq} = \big(\exp \mathbf{A}\big)_{pq}$
is the so-called *communicability* between nodes p and q,
$A_{pq} = \begin{cases} 1, & \text{if } \rho \geq \rho_c \\ 0, & \text{otherwise} \end{cases}$
is the adjacency matrix induced by the distance correlation matrix Cor_{ij}, and $\mathbf{E}(r)$ is a matrix such that when added to $\mathbf{A}$, yields a new graph $G(r) = (V, E')$ with all edges connecting $r \in V$ removed. The constant $C = (n-1)^2 - (n-1)$ normalizes ω_r such that it takes values between 0 and 1. We can better understand what ω_r is counting by re-writing the matrix exponential as a taylor series:
$\exp \mathbf{A} = \sum_{k=1}^{\infty} \frac{\mathbf{A}^k}{k!}$
Rasing the adjacency matrix to the power of k counts all walks from p to q of length k. The matrix exponential therefore counts all possible ways of moving from p to q weighted by the inverse factorial of k. So the denominator of ω_r counts all weighted walks involving every node. Put simply,

$$\boxed{\text{Communicability Betweenees centrality} = \frac{\text{sum of all weighted walks involving node r}}{\text{sum of all weighted walks involving every node}}}$$

So the communicability betweenness centrality is proportional to the number of connections (correlations) a node has and therefore satisifies the first requirement of relative risk. Next, we explore how this measure quantifies the spread of impacts throughout the network, satisfying our second requirement.

3.1 The Physics of what Communicability Measures

Estrada & Hatano (2007) provided an ingenius argument showing the communicability of a network is identical to the Green's function of a network. That is, it measures how impacts (or more generally thermal fluctuations) travel from one node to another. Their argument works by treating each node as an oscillator and each edge as a spring (which is what we did to generate the visualization of our asset correlation network). Intuitively, we can draw an analogy between the movement of an asset's price and its motion in a ball-spring system. In this analogy, volatility is equivalent to how energetic the oscillator is. Revisiting the hypothetical scenerio of Apple losing 40% of its value: we can visualize this in our mind's eye as an impact to one of the masses—causing it to violently oscillate. How does this motion propagate throughout the rest of the ball-spring system? Which masses absorb the blow and which reflect it? Communicability betweeness centrality answers this question by counting all possible ways the impact can reach node r. Higher values indicate the node has a greater susceptiblility to impacts whereas lower values denote just the opposite.

4 The Bottom Line

The communicability of a network quantifies how impacts spread from one node to another. In the context of an asset correlation network, communicability measures how volatility travels node to node. **We aim to position our capital such that it's the most resistant to the communicability of volatility**. Recall we seek a portfolio that (1) consistently generates wealth while minimizing potential losess and (2) is robust against large market fluctuations and economic downturns. Of course, generous returns are desired, but not in a way that threatens our initial investment. To this end, the strategy moving forward is this: allocate capital inversely proportional to its relative (or intraportfolio) risk.

4.1 Intraportfolio Risk

In [28]:
```
zero_degree = []
nonzero_degree = []
for t, d in H_close.degree():
    if d == 0:
        zero_degree.append(t)
    else:
        nonzero_degree.append(t)

print(zero_degree)
print(len(zero_degree))
print(nonzero_degree)
print(len(nonzero_degree))
```

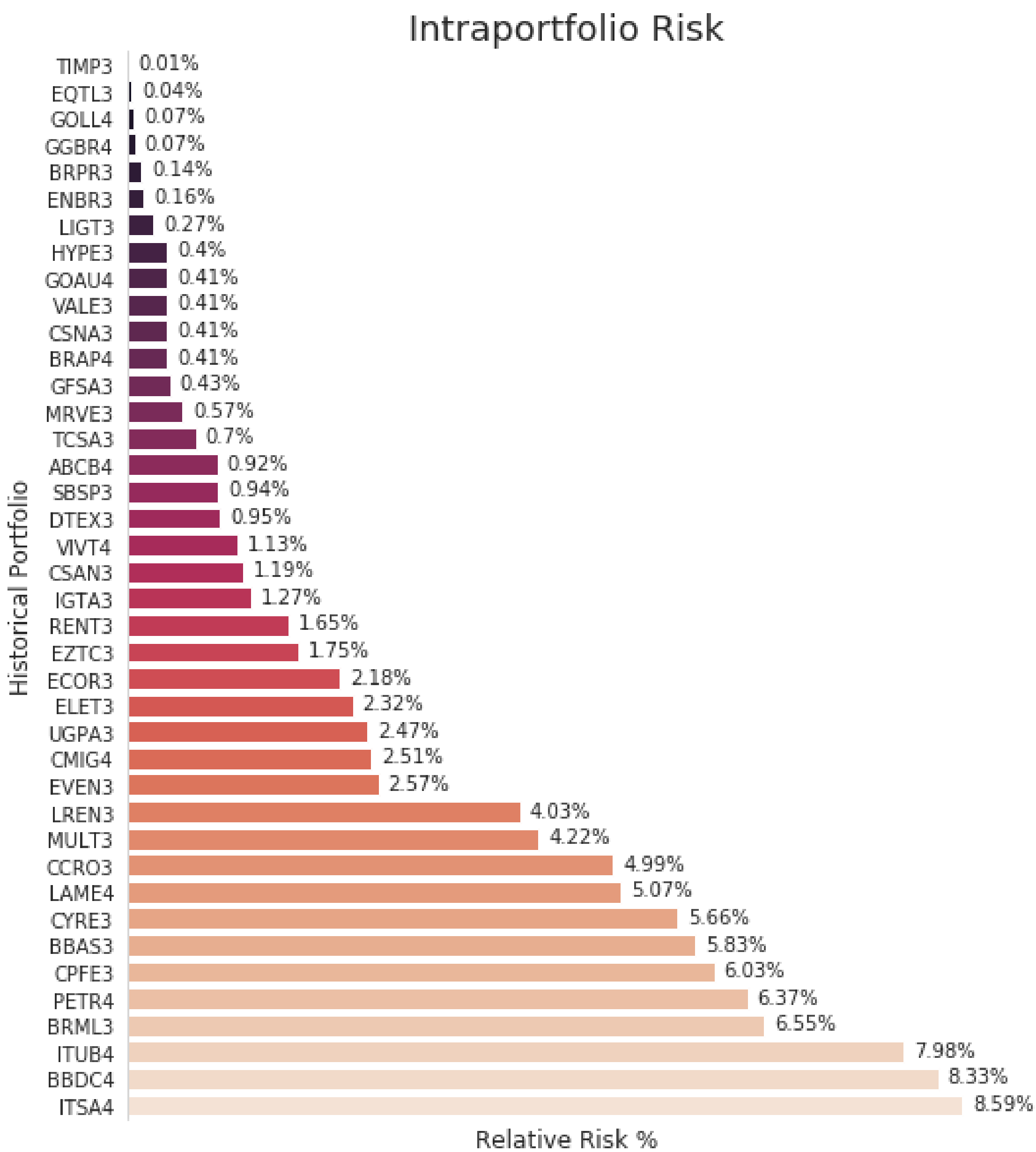
```
[]
0
['ABCB4', 'BBAS3', 'BBDC4', 'BRAP4', 'BRML3', 'BRPR3', 'CCRO3', 'CMIG4', 'CPFE3', 'CSAN3', 'CSI
40
```

In [29]:
```
len(zero_degree)
```

Out[29]:
```
0
```

In [30]:
```
# calculates the communicability betweeness centrality and returns a dictionary
risk_alloc = nx.communicability_betweenness_centrality(H_close)
#risk_alloc = nx.eigenvector_centrality(H_master)
# print(risk_alloc)
# converts the dictionary of degree centralities to a pandas series
risk_alloc = pd.Series(risk_alloc)

# normalizes the degree centrality
risk_alloc = risk_alloc / risk_alloc.sum()

# resets the index
risk_alloc.reset_index()
```

```python
# converts series to a sorted DataFrame
risk_alloc = (
    pd.DataFrame({"Stocks": risk_alloc.index, "Risk Allocation": risk_alloc.values})
        .sort_values(by="Risk Allocation", ascending=True)
        .reset_index()
        .drop("index", axis=1)
)

with sns.axes_style('whitegrid'):
    # initializes figure
    plt.figure(figsize=(8,10))

    # plots a pretty seaborn barplot
    sns.barplot(x='Risk Allocation', y='Stocks', data=risk_alloc, palette="rocket")

    # removes spines
    sns.despine(right=True, top=True, bottom=True)

    # turns xticks off
    plt.xticks([])

    # labels the x axis
    plt.xlabel("Relative Risk %", size=12)

    # labels the y axis
    plt.ylabel("Historical Portfolio", size=12)

    # figure title
    plt.title("Intraportfolio Risk", size=18)

    # iterates over the stocks (label) and their numerical index (i)
    for i, label in enumerate(list(risk_alloc.index)):

        # gets the height of each bar in the barplot
        height = risk_alloc.loc[label, 'Risk Allocation']

        # gets the relative risk as a percentage (the labels)
        label = (risk_alloc.loc[label, 'Risk Allocation']*100
                    ).round(2).astype(str) + '%'

        # annotates the barplot with the relative risk percentages
        plt.annotate(str(label), (height + 0.001, i + 0.15))
```




```
In [31]: # Helper to get company names
         df.loc[df.Ticker == "ITSA4"].Name.iloc[0]
```

```
Out[31]: 'ITAUSA'
```

We read an intraportfolio risk plot like this: VALE3 (*Companhia Vale do Rio Doce*) is $\frac{0.41}{0.07} = 5.86$ times riskier than GGBR4 (*Gerday*), BBDC4 (*Banco Bradesco*) is $\frac{8.33}{1.65} = 5.05$ times riskier than RENT3 (Localiza), ... , and ITSA4 (Itaú S.A.) is $\frac{8.59}{0.16} = 53.69$ times more risky than EMBR3 (Embraer)!

Intuitively, the assets that cluster in the center of the network are most susceptible to impacts, whereas those further from the cluster are the least susceptible. The logic from here is straightforward: take the inverse of the relative risk (which we call the "relative certainty") and normalize it such that it adds to 1. These are the asset weights. Formally,

$$\mathrm{w}_r = \frac{1}{\omega_r \sum_{r'} \omega_{r'}^{-1}}$$

Next, Let's visualize the allocation of 10,000 (USD) in our portfolio.

4.2 Communicability-Based Asset Allocation

In [32]:
```
# calculates degree centrality and assigns it to investmnet_A
investment_A = nx.communicability_betweenness_centrality(H_close)
#investment_A = nx.eigenvector_centrality(H_close)

# calculates the inverse of the above and re-asigns it to investment_A as a pandas se
investment_A = 1 / pd.Series(investment_A)

# normalizes the above
investment_A = investment_A / investment_A.sum()
investment_A
```

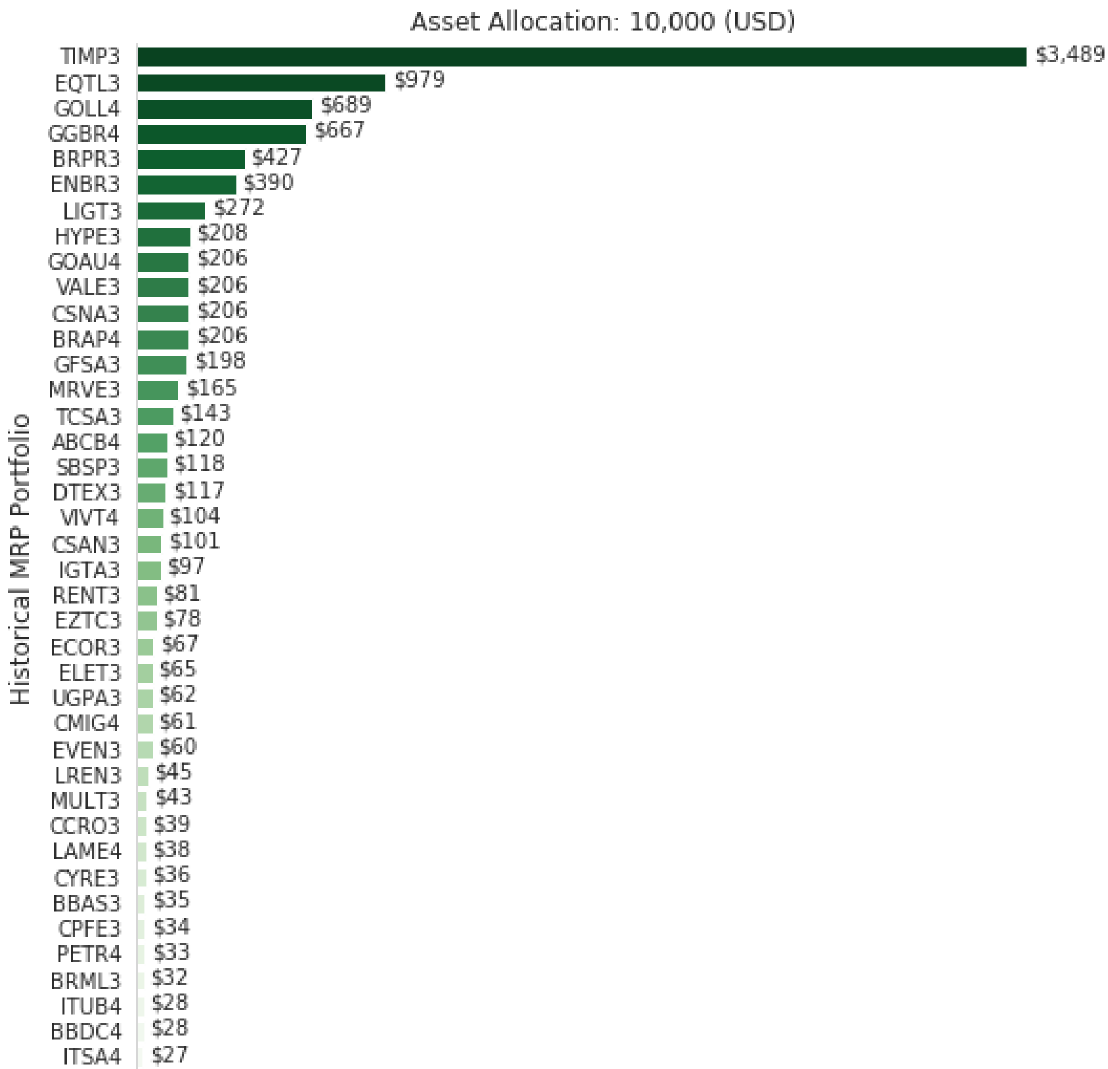
Out[32]:
```
ABCB4    0.003991
BBAS3    0.000630
BBDC4    0.000441
BRAP4    0.009008
BRML3    0.000561
BRPR3    0.026876
CCRO3    0.000736
CMIG4    0.001465
CPFE3    0.000609
CSAN3    0.003081
CSNA3    0.009008
CYRE3    0.000650
DTEX3    0.003854
ECOR3    0.001683
ELET3    0.001588
ENBR3    0.023474
EQTL3    0.093438
EVEN3    0.001428
EZTC3    0.002099
GFSA3    0.008530
GGBR4    0.052519
GOAU4    0.009008
GOLL4    0.055231
HYPE3    0.009154
IGTA3    0.002894
ITSA4    0.000428
ITUB4    0.000461
```

```
LAME4    0.000726
LIGT3    0.013704
LREN3    0.000912
MRVE3    0.006452
MULT3    0.000870
PETR4    0.000577
RENT3    0.002228
SBSP3    0.003931
TCSA3    0.005241
TIMP3    0.628771
UGPA3    0.001488
VALE3    0.009008
VIVT4    0.003249
dtype: float64
```

In [34]:
```
def softmax_temp(preds, temperature=1.0):
    # helper function to sample an index from a probability array
    # preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    return preds
```

In [35]:
```
# calculates degree centrality and assigns it to investmnet_A
investment_A = nx.communicability_betweenness_centrality(H_close)
#investment_A = nx.eigenvector_centrality(H_close)

# calculates the inverse of the above and re-asigns it to investment_A as a pandas se
investment_A = 1 / pd.Series(investment_A)

# normalizes the above
investment_A = softmax_temp(investment_A, 1.5)# np.exp(investment_A) / np.exp(investm

# resets the index
#investment_A.reset_index()

# converts the above series to a sorted DataFrame
investment_A = (
    pd.DataFrame({"Stocks": investment_A.index, "Asset Allocation": investment_A.value
        .sort_values(by="Asset Allocation", ascending=False)
        .reset_index()
        .drop("index", axis=1)
)

with sns.axes_style('whitegrid'):
    # initializes a figure
    plt.figure(figsize=(8,9))
```

```
# plot a pretty seaborn barplot
sns.barplot(x='Asset Allocation', y='Stocks', data=investment_A, palette="Greens_

# despines the figure
sns.despine(right=True, top=True, bottom=True)

# turns xticks off
plt.xticks([])

# turns the x axis label off
plt.xlabel('')

# fig title
plt.title("Asset Allocation: 10,000 (USD)", size=12)

# y axis label
plt.ylabel("Historical MRP Portfolio", size=12)

# captial to be allocated
capital = 10000

# iterates over the stocks (label) and their numerical indices (i)
for i, label in enumerate(list(investment_A.index)):

    # gets the height of each bar
    height = investment_A.loc[label, 'Asset Allocation']

    # calculates the capital to be allocated
    label = (investment_A.loc[label, 'Asset Allocation'] * capital
            ).round(2)

    # annotes the capital above each bar
    plt.annotate('${:,.0f}'.format(label), (height + 0.002, i + 0.15))
```



In [36]:
```
investment_A.iloc[:, 1].cumsum()
```

Out[36]:
```
0     0.348924
1     0.446817
2     0.515766
3     0.582438
4     0.625093
5     0.664069
6     0.691294
7     0.712097
8     0.732680
9     0.753263
10    0.773845
11    0.794428
```

```
12    0.814276
13    0.830753
14    0.845098
15    0.857060
16    0.868901
17    0.880586
18    0.891014
19    0.901081
20    0.910735
21    0.918843
22    0.926637
23    0.933363
24    0.939833
25    0.946028
26    0.952160
27    0.958188
28    0.962658
29    0.966991
30    0.970867
31    0.974706
32    0.978272
33    0.981767
34    0.985183
35    0.988477
36    0.991711
37    0.994547
38    0.997301
39    1.000000
Name: Asset Allocation, dtype: float64
```

In [37]:
```
investment_A.iloc[:10]
```

Out[37]:

	Stocks	Asset Allocation
0	TIMP3	0.348924
1	EQTL3	0.097893
2	GOLL4	0.068949
3	GGBR4	0.066672
4	BRPR3	0.042655
5	ENBR3	0.038976
6	LIGT3	0.027225
7	HYPE3	0.020803
8	GOAU4	0.020583
9	VALE3	0.020583

In [38]:
```
investment_A.tail()
```

Out[38]:

	Stocks	Asset Allocation
35	PETR4	0.003294
36	BRML3	0.003235

```
37   ITUB4          0.002835
38   BBDC4          0.002754
39   ITSA4          0.002699
```

It's worth pointing out that the methods we've used to generate the asset allocation weights differ dramatically from the contemporary methods of MPT and its extensions. The approach taken in this project makes no assumptions of future outcomes of a portfolio, i.e., the algorithm doesn't require us to make a prediction of the expected returns (as MPT does). What's more—we're not solving an optimization problem—there's nothing to be minimized or maximized. Instead, we observe the topology (interrelatedness) of our portfolio, predict which assets are the most susceptible to the communicability of volatile behaviour and allocate capital accordingly.

In [39]:
```python
df_train_close_copy.index.max()
```

Out[39]:
```
'2016-12-29'
```

In [40]:
```python
# DataFrame of the prices we buy stock at
df_buy_in = df_train_close_copy.loc[TRAIN_RANGE[1]].sort_index().to_frame('Buy In: {}
df_buy_in
```

Out[40]:
```
        Buy In: 2016-12-29
Ticker
ABCB4                13.93
BBAS3                28.09
BBDC4                29.00
BRAP4                14.85
BRML3                11.95
BRPR3                 7.50
CCRO3                15.96
CMIG4                 7.71
CPFE3                25.21
CSAN3                38.15
CSNA3                10.85
CYRE3                10.27
DTEX3                 6.80
ECOR3                 8.24
ELET3                22.81
ENBR3                13.40
EQTL3                54.40
EVEN3                 3.70
EZTC3                15.65
GFSA3                 1.86
GGBR4                10.80
GOAU4                 4.80
GOLL4                 4.62
HYPE3                26.13
IGTA3                26.67
ITSA4                 8.28
ITUB4                33.85
```

```
LAME4          17.00
LIGT3          17.36
LREN3          23.17
MRVE3          10.94
MULT3          59.38
PETR4          14.87
RENT3          34.22
SBSP3          28.79
TCSA3           2.16
TIMP3           7.83
UGPA3          68.45
VALE3          25.68
VIVT4          44.08
```

4.3 Alternative Allocation Strategy: Allocate Capital in the Maximum Independent Set

The maximum independent set (MIS) is the largest set of vertices such that no two are adjacent. Applied to our asset correlation network, the MIS is the greatest number of assets such that every pair has a correlation below ρ_c. The size of the MIS is inversely proportional to the threshold ρ_c. Larger values of ρ_c produce a sparse network (more edges are removed) and therefore the MIS tends to be larger. An optimized portfolio would therefore correspond to maximizing the size of the MIS subject to minimizing ρ_c. The best way to do this is to increase the universe of assets we're willing to invest in. By further diversifying the portfolio with many asset types and classes, we can isolate the largest number of minimally correlated assets and allocate capital inversely proportional to their relative risk. While generating the asset weights remains a non-optimization problem, generating the asset correlation network *becomes* one. We're really solving two sepreate problems: determing how to build the asset correlation network (there are many) and determining which graph invariants (there are many) extract the asset weights from the network. As such, one can easily imagine a vast landscape of portfolios beyond that of MPT and a metric fuck-tonne of wealth to create. **Unfortunately, solving the MIS problem is NP-hard. The best we can do is find an approximation.**

4.4 Using Expert Knowledge to Approximate the Maximum Independent Set

We have two options: randomly generate a list of maximal indpendent sets (subgraphs of G such that no two vertices share an edge) and select the largest one, or use expert knowledge to reduce the number of sets to generate and do the latter. Both methods are imperfect, but the former is far more computationally expensive than the latter. Suppose we do fundamentals research and conclude ITUB4 must be in our portfolio. How could we imbue the algorithm with this knowledge? Can we make the algorithm flexible enough for portfolio managers to fine-tune with goold-ole' fashioned research, while at the same time keeping it rigged enough to prevent poor decisions from producing terribe portfolios? We confront this problem in the code block below by extracting an approximate MIS by generating 100 random maximal indpendent sets containing ITUB4.

In [41]:
```
# a function to generate a random approximate MIS
### WARNING: rerunning kernel will produce different MISs
from header import generate_mis
```

```
In [42]: max_ind_set = generate_mis(H_close, sample_size=500)
         print(max_ind_set)
```

```
['ELET3', 'BRPR3', 'GFSA3', 'CSAN3', 'EZTC3', 'IGTA3', 'GGBR4', 'LAME4', 'EQTL3', 'ENBR3', 'GOI
```

The `generate_mis` function generates a maximal independent set that approximates the true maximum independent set. As an option, the user can pick a list of assets they want in their portfolio and `generate_mis` will return the safest assets to complement the user's choice. Picking UNH and AMZN left us with VZ and MCD. The weights of these assets will remain directly inversely proportional to the communicability betweeness centrality.

```
In [43]: # prices of shares to buy for the MIS
         df_mis_buy_in = df_buy_in.loc[list(max_ind_set)]
         df_mis_buy_in
```

Out[43]:

Ticker	Buy In: 2016-12-29
ELET3	22.81
BRPR3	7.50
GFSA3	1.86
CSAN3	38.15
EZTC3	15.65
IGTA3	26.67
GGBR4	10.80
LAME4	17.00
EQTL3	54.40
ENBR3	13.40
GOLL4	4.62
HYPE3	26.13
TIMP3	7.83
DTEX3	6.80
TCSA3	2.16
ABCB4	13.93
RENT3	34.22
SBSP3	28.79

5 Backtesting with Modern Portfolio Theory

Now that we have a viable alternative to portfolio optimization, it's time to see how the Hedgecraft portfolio performed in the validation years (15', 16', and 17') with respect to the Markowitz portfolio (i.e., the efficient frontier model) and the overall market. To summarize our workflow thus far we:

1. Preprocessed historical pricing data of 31 stocks for time series analyses.
2. Computed the distance correlation matrix $\rho_D(X_i, X_j)$ for the `Open`, `High`, `Low`, `Close`, and `Close_diff` from 2006-2014.
3. Used the NetworkX module to transform each distance correlation matrix into a weighted graph.

4. Adopted the winner-take-all method by Tse, et al. and removed edges with correlations below a threshold value of $\rho_c = 0.325$.
5. Built a master network by averaging over the edge weights of the `Open`, `High`, `Low`, `Close`, and `Close_diff` networks.
6. Calculated the "relative risk" of each asset as the communicabality betweeness centrality assigned to each node.
7. Generated the asset weights as the normalized inverse of communicability betweeness centrality.

In addition to the above steps, we introduced a human-in-the-middle strategy, giving the user flexible control over the portfolio construction process. This is the extra step we added:

8. Adjust the asset weights for an approximate maximum independent set, either with or without human intervention.

To distinguish bewteen these two approaches we designate steps 1-7 as the *Hedgecraft* algo and steps 1-8 as the *Hedgecraft MIS* algo. Below we observe how these models perform with the Efficient Frontier as a benchmark.

5.1 Generating Hedgecraft Portfolio Weights

```
In [44]: # calculates communicability betweeness centrality
         weights = nx.communicability_betweenness_centrality(H_close)
         #weights = nx.eigenvector_centrality(H_master)

         # dictionary comprehension of communicability centrality for the maximum independent 
         mis_weights = {key: weights[key] for key in list(max_ind_set)}


         # a function to convert centrality scores to portfolio weights
         from header import centrality_to_portfolio_weights

         print(centrality_to_portfolio_weights(weights))
         print('\n')
         print(centrality_to_portfolio_weights(mis_weights))
```

```
{'ABCB4': 0.004, 'BBAS3': 0.001, 'BBDC4': 0.0, 'BRAP4': 0.009, 'BRML3': 0.001, 'BRPR3': 0.027,


{'ELET3': 0.002, 'BRPR3': 0.029, 'GFSA3': 0.009, 'CSAN3': 0.003, 'EZTC3': 0.002, 'IGTA3': 0.00
```

Hedgecraft MIS allocates a staggering 91.4% of the investment to UNH. At first sight this portfolio appears far less diversified than Hedgecraft. However, if we recall, the relative risk of UNH was two orders of magnitude smaller than **every** other security (with the sole exception of Altaba). If our only options are the above 31 securities, the algorithm predicts UNH is the safest pick and allocates accordingly.

5.2 Allocating Shares to the Hedgecraft Portfolio

In [ ]:
```
# !pip install pyportfolioopt
```

In [45]:
```
# imports a tool to convert capital into shares
from pypfopt import discrete_allocation

# returns the number of shares to buy given the asset weights, prices, and capital to
alloc = discrete_allocation.DiscreteAllocation(
    weights,
    df_buy_in['Buy In: {}'.format(TRAIN_RANGE[1])],
    total_portfolio_value=capital
)

# returns same as above but for the MIS
mis_alloc = discrete_allocation.DiscreteAllocation(
    mis_weights,
    df_mis_buy_in['Buy In: {}'.format(TRAIN_RANGE[1])],
    total_portfolio_value=capital
)
```

```
33 out of 40 tickers were removed
12 out of 18 tickers were removed
```

In [46]:
```
alloc = alloc.greedy_portfolio()[0]

mis_alloc = mis_alloc.greedy_portfolio()[0]
```

In [47]:
```
# converts above shares to a pandas series
alloc_series = pd.Series(alloc, name='Shares')

# names the series
alloc_series.index.name = 'Assets'

# resets index, prints assets with the shares we buy
alloc_series.reset_index
print(alloc_series)

print('\n')

# does same as above but for the MIS
mis_alloc_series = pd.Series(mis_alloc, name='MIS Shares')
mis_alloc_series.index.name = 'Assets'
mis_alloc_series.reset_index
print(mis_alloc_series)
```

```
Assets
TIMP3    803
```