Improved Portfolio Diversification Through Unsupervised Learning

Michael J. Lewis

Original version: August 6, 2019

This version: August 6, 2023

Abstract

This paper introduces the Recursive Clustering Risk Parity (RCRP) approach. The RCRP method builds on the philosophy first introduced in Hierarchical Risk Parity (HRP) in [1], leveraging unsupervised learning to recursively build an optimal portfolio.

HRP established the concept of building a diversified portfolio using the inherent structure of the covariance matrix, utilizing graph theory and hierarchical clustering. In doing so, HRP avoided inverting the covariance matrix, a numerically unstable procedure when performed on the notoriously ill-conditioned, if not singular, empirical covariance matrix in financial applications. However, the procedure relies on sorting the underlying instruments via their relationships and, in doing so, creating a quasi-diagonalization of the matrix, but subsequently forgets those relationships. RCRP builds on this premise by utilizing this underlying tree structure of relations, leveraging improved clustering techniques and inversion approximations to enhance overall portfolio performance.
Keywords: Risk parity, tree graph, cluster, recursive, inverse rank-one update, metric space.

JEL Classification: G0, G1, G2.

AMS Classification: 91G10, 91G60, 91G70, 60E.

1 Portfolio Optimization: A Background

This paper proposes a method to obtain two desirable portfolios: the minimum variance portfolio and the maximum sharpe ratio portfolio. We consider both the case where there are limits on shorts, and when there are not; initially, we will focus on the case without limits.

For completeness, a portfolio is a vector of weights w across N market instruments. When considering the case of no shorting limits, the weights sum to unity, or $1^T w = 1$, while in the case with limits the absolute weights sum to unity, or $\sum_i |w_i| = 1$. Given the covariance matrix Σ, expected excess returns vector μ, and portfolio weights w, the variance of the portfolio is $w^T \Sigma w$ and the expected excess return of the portfolio is $w^T \mu$. With that terminology in place, we will be discussing the minimum variance portfolio without shorting limits, which satisfies

$$\begin{aligned} \arg\min_w \quad & w^T \Sigma w \\ \text{s.t.} \quad & 1^T w = 1 \end{aligned} \tag{1}$$

and the maximum sharpe ratio portfolio without shorting limits, which satisfies

$$\begin{aligned} \arg\max_w \quad & \frac{w^T \mu}{\sqrt{w^T \Sigma w}} \\ \text{s.t.} \quad & 1^T w = 1 \end{aligned} \tag{2}$$

The known optimal solution to equation (1) is setting $w = \frac{\Sigma^{-1} 1}{1^T \Sigma^{-1} 1}$, or equivalently $w \propto \Sigma^{-1} 1$, whereas the known optimal solution to equation (2) is setting $w \propto \Sigma^{-1} \mu$.[1] Put more concisely, the solutions set $w \propto \Sigma^{-1} b$, where $b = 1$ for the minimum variance portfolio, and $b = \mu$ for the max sharpe ratio. Unfortunately, both solutions require inverting the covariance matrix, a numerically unstable task when it is possible, and impossible when the covariance matrix

[1] See Appendix A for details

is singular, as is often the case in practice when involving thousands of stocks. Aside from this inherent numerical instability, empirical covariance matrices can themselves be noisy when evaluated on a finite number of financial returns, further complicating matters. A method that is robust to noise and does not require ill-conditioned matrix inversion is desirable.

One notable option for the minimum variance portfolio is the inverse variance portfolio (IVP). In this case, the weight w_i given to instrument i is set to be inversely proportional to its variance σ_i^2, or $w_i \propto \frac{1}{\sigma_i^2}$. Observe that this is the optimal solution in the special case where the instruments are independent, or equivalently $\Sigma = \Sigma_{diag}$ is a diagonal matrix with $\sigma_{ij} = \Sigma_{ij} = 0$ for $i \neq j$. HRP notably uses this result when recursively evaluating the portfolio weights subsequent to quasi-diagonalizing the covariance matrix. However, in finance it is often the case that instruments are not independent, and for this reason the inverse variance portfolio is suboptimal.

2 Improved Inverse Approximation

A notable improvement to the inverse variance portfolio can be found in [2]. We assume a constant correlation ρ between each instrument i, j with $i \neq j$, then $\sigma_{ij} = \rho\sigma_i\sigma_j$, and thus

$$\Sigma = (1-\rho)\Sigma_{diag} + \rho\sigma\sigma^T$$

where I is the identity matrix, and σ is the column vector of standard deviations. Given this is a rank-one update to the matrix $(1-\rho)\Sigma_{diag}$, we can use the Sherman-Morrison Identity to evaluate its analytic inverse.

$$\Sigma^{-1} = \frac{1}{1-\rho}\Sigma_{diag}^{-1} - \frac{\rho}{(1-\rho)(1+\rho(N-1))}\left(\frac{1}{\sigma}\right)\left(\frac{1}{\sigma}\right)^T$$

For a covariance matrix with this form, the optimal minimum variance port-

folio takes the form

$$w_i \propto \frac{1}{\sigma_i^2} - \frac{\rho \sum_j \frac{1}{\sigma_j}}{(1+\rho(N-1))\sigma_i}$$

while the optimal maximum sharpe ratio portfolio has

$$w_i \propto \frac{\mu_i}{\sigma_i^2} - \frac{\rho \sum_j \frac{\mu_j}{\sigma_j}}{(1+\rho(N-1))\sigma_i} \propto \frac{1}{\sigma_i}\left((1-\rho)s_i + \rho N(s_i - \bar{s})\right)$$

where $s_i = \frac{\mu_i}{\sigma_i}$ is the ith sharpe ratio, and $\bar{s}$ is the average sharpe ratio over the N instruments. This predictably reduces to the inverse variance portfolio in the case where $\rho = 0$. In practice, we can set $\rho = \bar{\rho}$, the average off-diagonal correlation, if we wish to approximate the covariance matrix this way.

While the above enhancement is a useful approximation in cases where the off-diagonal correlations are all similar, this isn't the typical case. This is, however, a useful intermediate result, and the reader is advised to keep this in mind as we describe the general method in the next section.

3 Portfolio Enhancement Via Recursive Clustering

Consider a portfolio of stocks that are exclusively in the Industrial or the Technology sector. We expect Industrial stocks to have returns similar to other Industrial stocks, and likewise Technology stocks returns similar to other Technology stocks. However, Industrial stock performance will be less similar to Technology stocks. If we sort the indices to place the Industrials first followed by the Technology stocks, we should expect the correlation matrix to have a block-like formation. See Figure 1 for such an example of a hypothetical correlation matrix. Consider first the portfolio of just Industrials as portfolio 1, and the portfolio of just Technology stocks as portfolio 2. We could optimize

portfolio 1, then do similarly for portfolio 2, and finally optimize this portfolio of optimized portfolios. While technically suboptimal, its performance would likely be near optimal.

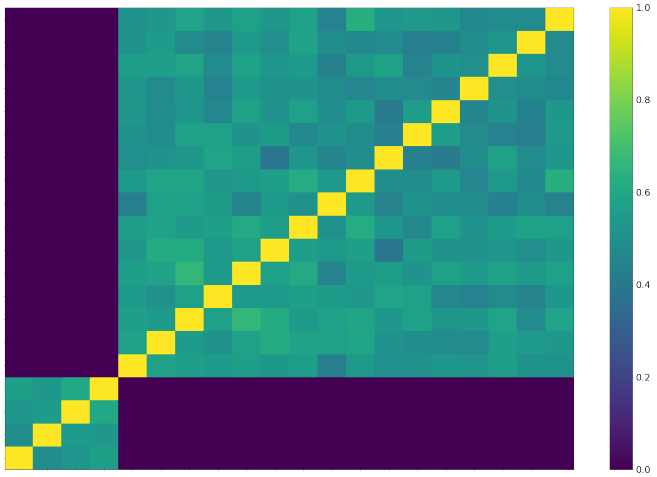

Figure 1: Random Correlation Matrix with 2 blocks

This thought experiment brings up a few discussion questions:

1. Some Industrial stocks perform more similarly to Technology stocks than other Industrials, so why delineate by sector?

2. Within the Technology sector, stocks A and B may perform more similarly to one another and less like stock C, so why not further dissect the stocks within each sector?

3. Why limit the analysis to a portfolio of just Industrials and Technology stocks?

For a more general portfolio, we will generalize this optimization procedure We

refer to this generalized optimal portfolio procedure as Recursive Clustering Risk Parity (RCRP), which takes as inputs a covariance matrix Σ of returns, and vector b. In the case of the minimum variance portfolio $b = 1$, and $b = \mu$ in the max sharpe ratio portfolio. The procedure can be described as follows:

1. Form the correlation matrix C from the covariance matrix Σ for a certain set of instruments S.

2. Use advanced unsupervised learning techniques to cluster, and partition S into some optimal k disjoint sets of instruments S_i.

3. For each set S_i, extract the covariance matrix Σ_i and vector b_i associated with those instruments.

4. If Σ_i is too small for clustering, find the optimal portfolio vector w_i via the improved approximate inverse portfolio optimization above using Σ_i and b_i. Otherwise, recursively set $w_i = RCRP(\Sigma_i, b_i)$. Note that w_i has dimension $N_i = |S_i|$ and $1^T w_i = 1$.

5. Using w_i to define the change in basis elements,

   (a) create the compressed covariance matrix Σ^{comp} and compressed vector b^{comp} for our portfolio of portfolios. This is equivalent to treating the performance of the optimized portfolios as individual instruments. Observe that $b_i^{comp} = b_i^T w_i$ and $\Sigma_{ij}^{comp} = w_i^T \Sigma_{ij} w_j$, where Σ_{ij} is the submatrix of Σ with rows related to instruments in S_i and columns related to instruments in S_j.

   (b) Find the cluster weights vector v via the improved inverse portfolio optimization using Σ^{comp} and b^{comp}. Note that v has dimension k, and $1^T v = 1$.

6. For $i = 1, \ldots, k$, set $w_i \to v_i w_i$, then concatenate the weight vectors w_i into a single vector w, and return this w as output. Note that $1^T w = 1$ as desired.

For step (2), we consider the ONC algorithm formulated in [2], though this could be enhanced should improved clustering techniques arise.

This technique reorganizes the index companies as a b-tree of relations, where companies that are highly correlated are "near" one another on the tree, while companies that are less correlated are further apart. This is consistent with the original premise of HRP, but unlike HRP fully utilizing risk parity at all levels of the tree. Furthermore, each non-leaf of the tree does contain 2 or more branches and does not assume equal-sized clusters, thus making it more generalized than the HRP procedure.

4 Numerical Examples

We now review the financial performance of RCRP. For this section, we consider an assortment of industry standard techniques for creating portfolio weights for the minimum variance portfolio in the no shorting limits case, i.e. where $1^T w = 1$, and compare the performance of the resulting portfolios from these techniques. The code for this section, written in Python 3, can be found at: `https://github.com/mlewis1729`, as well as in Appendix C. For transparency, there are two modifications made in the code from the source material:

1. HRP has been ported from Python 2 to Python 3 code using minor syntax modifications

2. The ONC algorithm from [2] has been

   (a) ported from Python 2 to Python 3 code using minor syntax modifications

(b) modified in the re-clustering step; where originally the clusters with better than average silhouette t-stats were not re-clustered recursively, now only the cluster with the highest silhouette score t-stat is not re-clustered. This slows calculation performance, but ideally improves overall clustering results.

Furthermore, for this section we define the effective number of positions $N_{eff} = \frac{1}{HHI} = \frac{1}{\sum_i w_i^2}$, where HHI is the Herfindahl–Hirschman index[2]. This metric evaluates how concentrated a portfolio is, as well as the effective number of data points in an exponentially weighted series. For example, using weights $w_i = (1-\alpha)\alpha^i$ gives us

$$N_{eff} = \frac{1}{\sum_i w_i^2} = \frac{1-\alpha^2}{(1-\alpha)^2}.$$

Given a halflife H, we have

$$\alpha = \exp(-\epsilon) \Rightarrow \alpha^H = \exp(-H\epsilon) = \frac{1}{2} = \exp(-\ln 2) \Rightarrow \epsilon = \frac{\ln 2}{H}.$$

This yields $N_{eff} \approx \frac{1-(1-2\epsilon)}{(1-(1-\epsilon))^2} = \frac{2}{\epsilon} = \frac{2H}{\ln 2}$.

The portfolio techniques we consider in this analysis are the following:

- RCRP, utilizing the extended terms from the inverse variance approximation (where $\rho = \bar{\rho}$) in step 5
- RCRP, without using the extended terms (equivalent to setting $\bar{\rho} = 0$) in step 5
- HRP
- the inverse variance portfolio, IVP,

[2] https://en.wikipedia.org/wiki/Herfindahl%E2%80%93Hirschman_index#Effective_assets_in_a_portfolio

- Using Ledoit-Wolf (LW) shrinkage on the empirical covariance matrix
- Using OAS shrinkage on the empirical covariance matrix

The scenario analyzed is as follows: For a given year, e.g. 2020, we fetch the daily returns for a large set of NYSE companies from 3 years back, e.g. 2017, to present. On the first day of each month in the chosen year, we use the historical returns up to the end of the previous month to create an empirical covariance matrix. In the cases of RCRP, HRP, and IVP we create an exponentially-weighted (EW) covariance matrix with a halflife of 60 days, while in the cases of the shrinkage matrices, which aren't coded to handle EW calculations in the standard libraries in Python, we utilize the entire 3+ years of daily returns. We then create the optimized portfolio weights using the above techniques, and with those portfolios evaluate the realized returns for that given month from the first to the end of that month. Note that the weights are not allowed to drift, but given these weights are held for a month, the impact from drift should be minimal. This procedure was run over recent years, specifically 2019-2022.

Figure 2 shows the EW moving standard deviation, halflife 20 days, of the realized returns given the above procedures. Figure 3 shows the same results but normalized by the result for RCRP using the extended terms so as to show relative performance. It is readily observable that IVP has the largest volatility, typically performing 2-3x and occasionally 4x worse (average 2.36x) than RCRP. HRP performs better than IVP but still typically 1.5x-2.5x worse (average 1.8x) than RCRP. The version of RCRP without the extended terms performs slightly worse (1-2x, average 1.35x) than the version with. Notably the shrinkage matrix methodologies LW and OAS have more comparable performance (0.5x-1.5x, average 1.2x) to RCRP; this may be due to the fact the covariance matrices they use have 3 years ($\tilde{7}50$ data points) worth of data, while RCRP is effectively using $\frac{2 \cdot 60}{\ln 2} \approx 174$ data points, a byproduct of using exponential weights with

halflife 60.

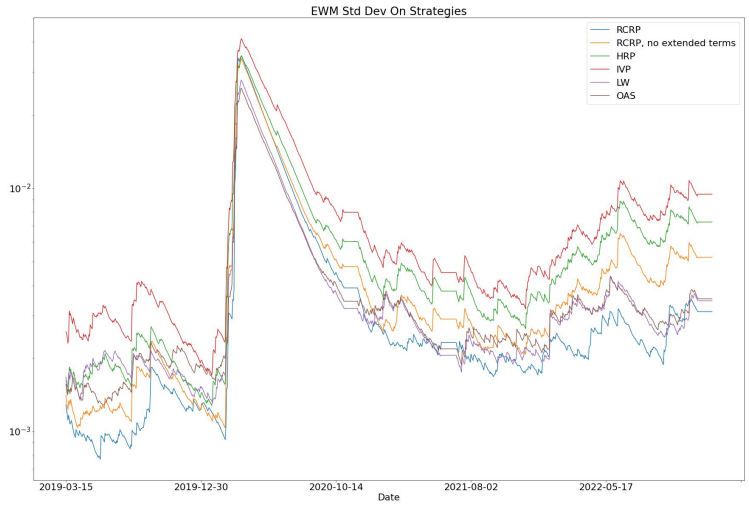

Figure 2: EW moving standard deviation of daily returns of strategies, HL=20

Given the portfolio weights, Figure 4 shows the effective number of positions in each portfolio over time. From this we see that IVP has the largest effective number of positions ($\sim$ 480), and thus lowest concentration in the portfolio, while HRP comes in second with a slightly lower effective position size ($\sim$ 200). The shrinkage matrix techniques come in the middle of the pack (LW $\sim$ 130, OAS $\sim$ 60) while RCRP with extended terms comes in the most concentrated at $\sim$ 20 positions. RCRP without the extended terms shows up in between the shrinkage matrix techniques with roughly 75 positions on average. An alternative metric for analyzing portfolio concentration is the maximum absolute weight at any given time. This is shown in Figure 5. As with Figure 4, we observe RCRP with extended terms is the most concentrated, with the largest position typically around 14%, while RCRP without the extended terms is typ-

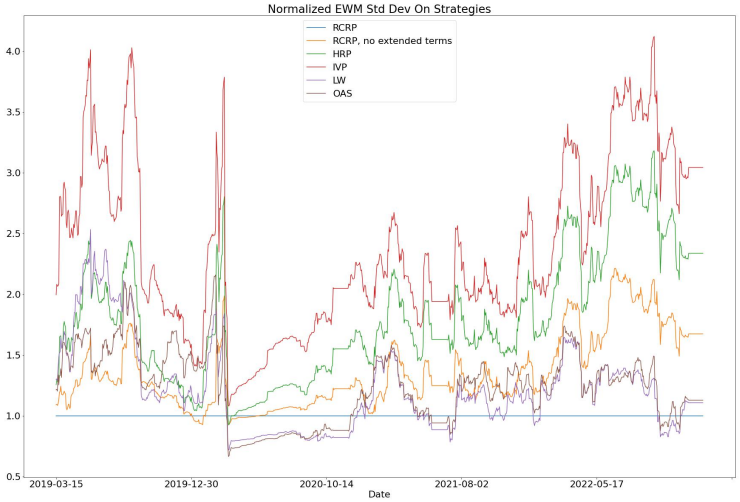

Figure 3: Normalized EW moving standard deviation of daily returns of strategies, HL=20

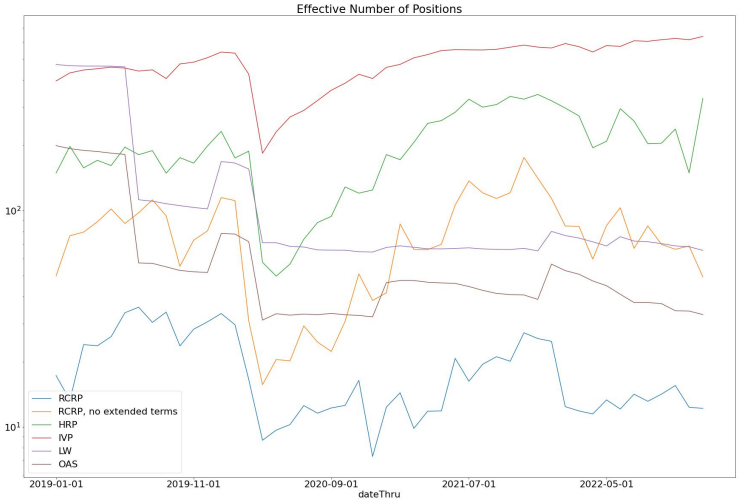

Figure 4: Effective Number of Positions

ically half that ($\sim 7\%$). HRP comes in third at roughly 3.2%, while LW, OAS, and IVP hold positions no larger than 1-2% on average. For completeness, we

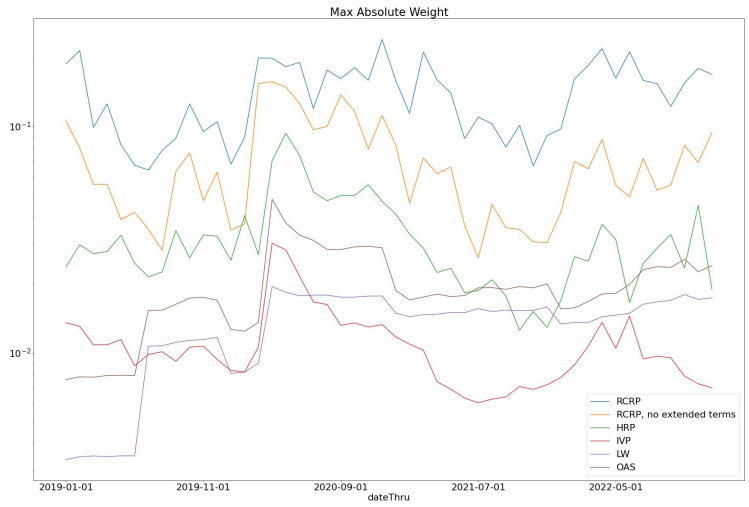

Figure 5: Max Absolute Weight

also include a chart with total number of positions, which is to say number of positions where weights are non-zero. This is shown in Figure 6. For this analysis, we are typically considering 1000-1600 companies.

5 Extended Commentary

5.1 Incorporating Limits On Short Positions

The above analysis and procedure focused on the scenario where there are no shorting limits. In the case of shorting limits, the above process is easily altered to incorporate this modification; we need only modify step (5b). See Appendix B for proof of the necessary procedure modification.

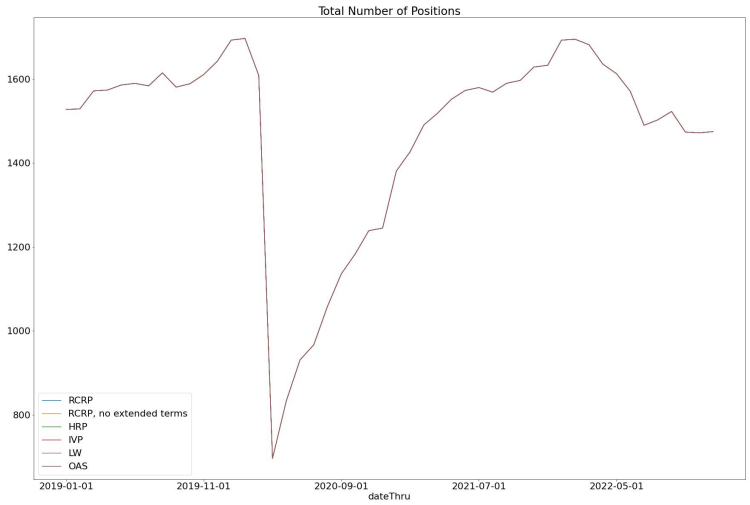

Figure 6: Total Number of Positions

5.2 Mathematical Justification For Design of RCRP

Let r denote the sampled correlation between two stocks, while ρ denotes the true correlation. It is known that the sampling standard deviation $\sqrt{\text{Var}(r)} = \frac{1-\rho^2}{\sqrt{n(1+\rho^2)}}$,[3] and thus the error bars on the sampled correlation are smaller when r is "high", or near ± 1, while the error bars are larger when r is "low", or near 0. RCRP by design treats stocks that are "near" each other, where r is high, more accurately by solving the optimal portfolio accurately near the leaves of the b-tree. Stocks that are "far" from each other, where r is lower, evaluate their weights at the more aggregate scale with portfolios of portfolios and the correlations are averaged across portfolios. By this interpretation, RCRP respects ′local" relationships, where the error bars are smaller between stocks, more than those that are further apart and thus have larger error bars. This may help in more accurately representing true relationships in the optimal portfolios.

[3]See `https://stats.stackexchange.com/questions/226380/derivation-of-the-standard-error-for-pearsons-correlation-coefficient`

5.3 Notable Alternative Use Cases

Section 3 describe how RCRP evaluates $w \propto \Sigma^{-1}\mu$. This is, up to scaling λ, equivalent to solving $\lambda\Sigma w = \mu$. This procedure could be easily modified for use in matrix multiplication by the inverse of the covariance matrix Σ. For example, each column i of $\Sigma^{-1}A$ equals $\Sigma^{-1}A_i$, where A_i is the ith column of A. Similarly, each row i of $B\Sigma^{-1}$ equals $B_i^T\Sigma^{-1}$, where B_i is the ith row of B. This is true due to the symmetrical nature of Σ. However, more consideration is necessary for deciding the appropriate λ in such scenarios.

6 Conclusions

This new procedure RCRP is designed to perform optimal portfolio calculations without the need for the often numerically unstable procedure of inverting the covariance matrix. RCRP builds on the original principles of HRP, while ensuring the original correlations of the companies are adhered to more thoroughly by designing a b-tree of relationships between the companies.

We observe that RCRP more effectively lowers the volatility of a portfolio than HRP or IVP, and even on average performs better than standard shrinkage matrix methodologies despite having significantly fewer data points. One notable concern is an increased concentration in the portfolio, a concern that is somewhat mitigated by not using the extended terms in the inverse approximating step, but which comes at the cost of reduced performance. Further testing would be necessary to better delineate this tradeoff.

A Optimal Max Sharpe Portfolios

In [1], a proof is given for the optimal minimum variance portfolio construction. The following proof for the optimal max sharpe portfolio construction is given

for completeness. A similar proof is given in [3].

Our goal is to solve

$$\begin{aligned} \arg\max_{w} \quad & \frac{w^T\mu}{\sqrt{w^T\Sigma w}} \\ \text{s.t.} \quad & 1^T w = 1 \end{aligned}$$

Let

$$\begin{aligned} g_1(w) &= w^T\mu \\ g_2(w) &= w^T\Sigma w \\ h(x_1, x_2) &= \frac{x_1}{\sqrt{x_2}} \\ f(w) &= \frac{w^T\mu}{\sqrt{w^T\Sigma w}} = h(g_1(w), g_2(w)) \end{aligned}$$

We set the derivative of $f(w)$ to 0, and find

$$\frac{1}{\sqrt{w^T\Sigma w}}\mu_i - \frac{w^T\mu}{2(w^T\Sigma w)^{3/2}}(2\Sigma w)_i = 0 \ \forall i \Rightarrow \Sigma w = \frac{w^T\Sigma w}{w^T\mu}\mu = C\mu$$

implying the optimal portfolio $w = C\Sigma^{-1}\mu$. Observe that $f(w) = f(Cw)$ for $C > 0$; thus, the optimal max sharpe portfolio has $w \propto \Sigma^{-1}\mu$. Note that this same argument extends to the case where short positions are limited, in which case the condition $\sum_i |w_i| = 1$ holds. Thus, the max sharpe portfolio with no shorting limits differs from the case with limits only by the normalization.

B Optimal Min Variance Portfolios With Shorting Limits

We now discuss the scenario where short positions on the portfolio are limited. To do so, we exchange the condition $\sum_i w_i = 1$ with $\sum_i |w_i| = 1$. Section A makes clear the max sharpe ratio portfolio with shorting limits differs only in normalization from the case without limits. In this section, we focus our

attention on the minimum variance portfolio.

We consider the problem

$$\begin{aligned} \arg\min_{w} \quad & w^T \Sigma w \\ \text{s.t.} \quad & \sum_i |w_i| = 1 \end{aligned} \tag{3}$$

Using Lagrange multipliers, we set the derivative of $w^T \Sigma w + \lambda(\sum_i |w_i| - 1)$ to 0, and find

$$\Sigma w = Cs \Rightarrow w = C\Sigma^{-1} s,$$

where $s_i = \sigma(w_i)$, the sign of w_i, and C is a normalizing constant. Observe that $s^T w = \sum_i |wi| = 1 = Cs^T \sigma^{-1} s$, and thus $C = \frac{1}{s^T \Sigma^{-1} s}$. Therefore, our optimal minimum variance portfolio is $w = \frac{\Sigma^{-1} s}{s^T \Sigma^{-1} s}$, and the minimum variance is $w^T \Sigma w = C^2 s^T \Sigma^{-1} \Sigma \Sigma^{-1} s = C$.

The difficulty of this problem depends on discovering the appropriate s. However, interpreting C is far simpler, as it is the smallest variance it can be, so we focus our attention on that. Let $S = \{s | s_i \in \{-1, 1\} \ \forall i\}$ be the set of all possible sign vectors, and thus $s \in S$. Thus, our problem reduces to finding

$$\arg\max_{s \in S} s^T \Sigma^{-1} s.$$

This is challenging to analyze for general covariance matrix Σ, so we reduce our consideration to the approximate matrix listed earlier, namely

$$\Sigma = (1 - \rho)\Sigma_{diag} + \rho \sigma \sigma^T.$$

Note that Σ is positive semi-definite. By the Matrix determinant lemma,

$$\det(\Sigma) = \det((1-\rho)\Sigma_{diag}) \left(1 + \frac{\rho\sigma^T\Sigma_{diag}^{-1}\sigma}{1-\rho}\right) \geq 0 \Rightarrow \frac{\rho\sigma^T\Sigma_{diag}^{-1}\sigma}{1-\rho} = \frac{N\rho}{1-\rho} \geq -1,$$

implying that $\rho \geq \frac{-1}{N-1}$ is a necessary condition. Next, observe that $s^T\Sigma_{diag}^{-1}s = \sum_i \frac{1}{\sigma_i^2} \forall s \in S$. Third, let $K = \frac{\rho}{1+\rho(N-1)}$, and thus $\sigma(K) = \sigma(\rho)$. Given that

$$\Sigma^{-1} = \frac{1}{1-\rho}\Sigma_{diag}^{-1} - \frac{\rho}{(1-\rho)(1+\rho(N-1))}\left(\frac{1}{\sigma}\right)\left(\frac{1}{\sigma}\right)^T$$

we find

$$s^T\Sigma^{-1}s = \frac{1}{1-\rho}\left(\sum_i \frac{1}{\sigma_i^2}\right) - \frac{K}{1-\rho}\left(\left(\frac{1}{\sigma}\right)^T s\right)^2$$

If $\rho < 0$, then $K < 0$, and thus maximizing $s^T\Sigma^{-1}s$ requires maximizing $|\left(\frac{1}{\sigma}\right)^T s|$. Given that $\sigma_i > 0 \ \forall i$, the optimal $s = \pm 1$. If $\rho > 0$, then $K > 0$, and thus maximizing $s^T\Sigma^{-1}s$ requires minimizing $|\left(\frac{1}{\sigma}\right)^T s|$.

Since s is a vector of $\pm$1s, we see that minimizing $|\left(\frac{1}{\sigma}\right)^T s|$ can be observed as the optimization version of the partition problem, though with real numbers rather than integers, which is known to be NP-hard [4]. For small N, we can check all $s \in S$, noting that $|S| = 2^N$. For improved computation, note that the variance is the same for $\pm s$; therefore, consider only s such that $s_1 = 1$, thereby reducing computation by a factor of 2. For larger N, an approximate solution must be found.

Consider the following greedy algorithm: let $s_1 = 1$, and let $y = \frac{1}{\sigma_i}$ hold the running sum. For $n > 1$, if $y > 0$, then set $s_n = -1$, and $y \leftarrow y - \frac{1}{\sigma_n}$; otherwise, set $s_n = 1$, and $y \leftarrow y + \frac{1}{\sigma_n}$. When $n = N$, observe that $y = \sum_j \frac{s_j}{\sigma_j}$. We can then repeat this greedy procedure multiple times by permuting the σ_i, running this procedure, performing the inverse permutation on the resulting s, and ultimately choose the s that obtains the smallest $|y|$.

Finally, there is the matter of confirming that, given the optimal s, the condition $s_i = \sigma(w_i)$ is satisfied. Given that $w = C\Sigma^{-1}s$, and $C > 0$, we observe

$$\begin{aligned} & w_i \propto \frac{s_i}{\sigma_i^2} - \frac{K}{\sigma_i}\sum_j \frac{s_j}{\sigma_j} \\ \Rightarrow & |w_i| = s_i w_i \propto \frac{1}{\sigma_i^2} - \frac{K s_i}{\sigma_i}\sum_j \frac{s_j}{\sigma_j} \geq 0 \\ \Rightarrow & \frac{1}{\sigma_i} - K s_i \sum_j \frac{s_j}{\sigma_j} \geq 0 \end{aligned}$$

is a necessary and sufficient condition. We see this condition is trivially satisfied when $\rho < 0 \Rightarrow K < 0$, with optimal $s = \pm 1$. Observe that $\rho \in [0, 1] \Rightarrow K \in \left[0, \frac{1}{N}\right]$. If $s_i \sum_j \frac{s_j}{\sigma_j} \leq 0$, then this condition is satisfied trivially; therefore, consider the case $s_i \sum_j \frac{s_j}{\sigma_j} \geq 0$. Observe that, if $s_i \geq 0$, then $\sum_j \frac{s_j}{\sigma_j} \geq 0$. In this scenario, if $\sum_j \frac{s_j}{\sigma_j} \geq \frac{1}{\sigma_i}$, the sum could be decreased by setting $s_i = -1$, and thus is not the optimal s. Consequently, in this scenario, $\sum_j \frac{s_j}{\sigma_j} < \frac{1}{\sigma_i}$, and thus the condition $K s_i \sum_j \frac{s_j}{\sigma_j} \leq \frac{1}{\sigma_i}$ is satisfied. A similarly argument confirms the case for $s_i < 0$, and thus we confirm the condition $s_i = \sigma(w_i)$ is satisfied for optimal s.

C Python Code

The section contains an implementation of RCRP, written in Python 3. It can also be found at: `https://github.com/mlewis1729`.

Note that the code below contains two separate implementations of the algorithm. One implementation is direct in that it allows a direct computation of weights, i.e.

```
weights = RCRP( cov )
```

The other implementation first constructs and saves the b-tree structure, which

can be time consuming. This structure can then be called multiple times with different vectors b, i.e.

```
Ct = Cov_Tree()
Ct.fit( cov )
weights = Ct.solve( b )
```

```
# On 20190731 by Michael Lewis <mjlewis@cims.nyu.edu>
# Recursive Clustering Risk Parity
import numpy as np, pandas as pd
import clusterCorr as cC
#------------------------------------------------------------------------------
# Recursively Clustered Risk Parity (RCRP)
#------------------------------------------------------------------------------
def compressCov(cov, b, clusters, w):
    # Takes a covariance matrix and reduces the it to the basis vectors
        made by the weights from the clusters
    # Similarly, reduces the vector b (e.g. None, or expected returns),
        and reduces by the cluster weights
    wM = pd.DataFrame(0., index=clusters.keys(), columns=cov.index )
    #print(clusters)
    for i in np.sort( list( clusters.keys() ) ):
        wM.loc[i, clusters[i] ] = w[clusters[i]]
    b_clustered = None if b is None else wM.dot( b )
    cov_clustered = wM.dot(cov).dot(wM.T)
    return cov_clustered, b_clustered
#------------------------------------------------------------------------------
def getIVPNew(cov, use_extended_terms=False, b=None, limit_shorts=False):
    # Compute the inverse variance portfolio via approximation of the
        covariance matrix
```

```
    n = int(cov.shape[0])
    corr = cC.cov2corr(cov)
    invSigma = 1./np.sqrt(np.diag(cov))
    if b is None:
        # If b is None, we're finding the min variance portfolio
        # If b is not None, we're finding the max sharpe portfolio => b
            = mu
        if limit_shorts and n > 1 and np.sum(np.sum(corr)) >= n:
            # if average correlation is >= 0 and limiting shorts (
                sum(|w|) = 1 )
            b = FindMinPermutation(invSigma)
        else:
            b = np.ones(n)
    ivp = b * (invSigma ** 2)
    if use_extended_terms and n > 1:
        # Obtain average off-diagonal correlation
        rho=(np.sum(np.sum(corr))-n)/(n**2-n)
        ivp-=rho*invSigma*np.sum(invSigma*b)/(1.+(n-1)*rho)
    if limit_shorts:
        # condition: sum(|w|) = 1
        return ivp / abs(ivp).sum()
    else:
        # condition: sum(w) = 1
        return ivp / ivp.sum()
#-------------------------------------------------------------------------------
def RCRP(cov, use_extended_terms=True, b=None, limit_shorts=False):
    # Create a risk balance portfolio by recursively clustering the
        correlation matrix,
    # using the inverse variance portfolio at the lowest levels, then
        utilizing the optimal weights at lower levels
```

```
# to compress the covariance matrix, and evaluating the optimal
    inverse variance portfolio on the compressed covariance matrix
# b = None => min variance portfolio, otherwise b = Expected returns

# default assume use extended terms
w = pd.Series( 1., index = cov.index )

# cluster the correlation matrix
_, clusters, _ = cC.clusterKMeansTop(cC.cov2corr( cov ))
if len( clusters.keys() ) > 1:
    for clusterId in clusters.keys():
        lbls = clusters[clusterId]
        if len(lbls) > 2:
            if b is None:
                w[lbls] = RCRP(cov.loc[lbls, lbls],
                    use_extended_terms=use_extended_terms,
                    limit_shorts=limit_shorts, b=None)
            else:
                w[lbls] = RCRP(cov.loc[lbls, lbls],
                    use_extended_terms=use_extended_terms,
                    limit_shorts=limit_shorts, b=b[lbls])
        else:
            if b is None:
                w[lbls] = getIVPNew(cov.loc[lbls, lbls],
                    use_extended_terms=use_extended_terms,
                    limit_shorts=limit_shorts, b=None)
            else:
                w[lbls] = getIVPNew(cov.loc[lbls, lbls],
                    use_extended_terms=use_extended_terms,
                    limit_shorts=limit_shorts, b=b[lbls].values)
```

```
            # compress the covariance matrix (and vector b) using the
                optimal weights from lower levels
            cov_compressed, b_compressed = compressCov(cov, b, clusters, w)

            # evaluate the inverse variance portfolio on the clustered
                porfolio
            if b is None:
                w_clusters = getIVPNew(cov_compressed,
                    use_extended_terms=use_extended_terms,
                    limit_shorts=limit_shorts, b=None)
            else:
                w_clusters = getIVPNew(cov_compressed,
                    use_extended_terms=use_extended_terms,
                    limit_shorts=limit_shorts, b=b_compressed.values)

            # update the weights using the optimal cluster weights
            for clusterId in clusters.keys():
                lbls = clusters[clusterId]
                w[lbls] *= w_clusters[clusterId]
        else:
            # Only has one cluster
            if b is None:
                w = getIVPNew(cov, use_extended_terms=use_extended_terms,
                    limit_shorts=limit_shorts, b=None)
            else:
                w = getIVPNew(cov, use_extended_terms=use_extended_terms,
                    limit_shorts=limit_shorts, b=b.values)
        return w
#-------------------------------------------------------------------------
def Int2Bin(n,L):
```

```
    # input: positive integer n < 2**L, positive integer L
    # output: string for n in base 2 of length L
    x = "{0:b}".format(n)
    return '0' * (L-len(x)) + x
#--------------------------------------------------------------------------
def Bin2Vector(x):
    # input: binary string x of length N, e.g. 01001
    # output: N vector v, e.g. np.asarray([-1 1 -1 -1 1])
    return np.asarray([2*int(b)-1 for b in x])
#--------------------------------------------------------------------------
def FindMinPermutation(invSigma):
    # Input: N vector containing 1/sigma > 0
    # output: N vector a s.t. sum( a * invSigma ) is minimized
    N = len(invSigma)
    maxN = 12
    a, minSum = None, np.inf
    if N <= maxN + 1:
        # N is small enough, just do the optimal value
        # NOTE: We don't need to check all 2^N possibilities; the sum
            for n2 = 2**N - 1 - n will have -sum of n
        for n in xrange(2**(N-1)):
            v = Bin2Vector(Int2Bin(n,N))
            s = abs(sum(v * invSigma))
            if s < minSum:
                a, minSum = v, s
    else:
        # N is sufficiently large, find an approximate solution
        X = pd.Series( np.copy(invSigma), index=range(N) )
        for _ in xrange(2**maxN):
            tmp = X.sample(frac=1)
```

```
            s = tmp.iloc[0]
            b = '1'
            for i in xrange(1,len(tmp)):
                if s > 0:
                    s -= tmp.iloc[i]
                    b += '0'
                else:
                    s += tmp.iloc[i]
                    b += '1'
            if abs(s) < minSum:
                minSum, a = abs(s), Bin2Vector( ''.join( [b[i] for i in
                     tmp.index.argsort()] ) )
    return a
#--------------------------------------------------------------------------------
# Build Class struct to hold B-Tree Structure, so structure can be
    re-used for different vectors
class Cov_Tree:
    def __init__( self, cluster_id = 0 ):
        self.cluster_id = cluster_id
        self.cluster_family_map = {}
        self.cluster_type_map = {}
        self.cov = None

    def __del__( self ):
        self.cluster_id = 0
        self.cluster_family_map = {}
        self.cluster_type_map = {}
        self.cov = None

    def fit( self, cov ):
```

```
# cluster the correlation matrix
_, clusters, _ = cC.clusterKMeansTop(cC.cov2corr( cov ))
main_id = self.cluster_id
if len( clusters ) > 1:
    this_id = main_id + 1
    cluster_set = []
    for clusterId in clusters.keys():
        lbls = clusters[clusterId]
        cluster_set.append( this_id )
        if len(lbls) > 2:
            Sub_Tree = Cov_Tree( cluster_id = this_id )
            Sub_Tree.fit( cov.loc[ lbls, lbls ] )
            for idx, lbls1 in Sub_Tree.cluster_family_map.items():
                self.cluster_family_map[ idx ] = lbls1
                self.cluster_type_map[ idx ] =
                    Sub_Tree.cluster_type_map[ idx ]
            this_id += len( Sub_Tree.cluster_family_map )
            del Sub_Tree
        else:
            # small emough to be a leaf
            self.cluster_family_map[ this_id ] = lbls
            self.cluster_type_map[ this_id ] = 'LEAF'
            this_id += 1
    self.cluster_family_map[ main_id ] = cluster_set
    self.cluster_type_map[ main_id ] = 'NODE'
else:
    clusterId = clusters.keys()[0]
    lbls = clusters[ clusterId ]
    self.cluster_family_map[ main_id ] = lbls
    self.cluster_type_map[ main_id ] = 'LEAF'
```

```
    self.cov = cov
    return self


def solve( self, b, use_extended_terms = True, limit_shorts = False
    ):
    # solve C x = b
    # b = None => min variance portfolio, otherwise b = Expected
        returns


    def recursive_solve( idx ):
        cluster_type = self.cluster_type_map[ idx ]
        lbls         = self.cluster_family_map[ idx ]
        if cluster_type == 'LEAF':
            sub_cov = self.cov.loc[ lbls, lbls ]
            if b is None:
                w = getIVPNew( sub_cov,
                    use_extended_terms=use_extended_terms,
                    limit_shorts=limit_shorts, b=None)
            else:
                sub_b   = b.loc[ lbls ]
                w = getIVPNew( sub_cov,
                    use_extended_terms=use_extended_terms,
                    limit_shorts=limit_shorts, b=sub_b.values)
            w = pd.Series( w, index = lbls )
            return w
        else:
            w = []
            clustering = {}
            for lbl in lbls:
                sub_w = recursive_solve( lbl )
```

```
            w.append( sub_w )
            clustering[ lbl ] = sub_w.index
        w = pd.concat( w )
        # compress the covariance matrix (and vector b) using the
            optimal weights from lower levels
        cov1 = self.cov.loc[ w.index, w.index ]
        b1 = None if b is None else b.loc[ w.index ]
        cov_compressed, b_compressed = compressCov( cov1, b1,
            clustering, w )

        # evaluate the inverse variance portfolio on the
            clustered porfolio
        if b is None:
            w_clusters = getIVPNew(cov_compressed,
                use_extended_terms=use_extended_terms,
                limit_shorts=limit_shorts, b=None)
        else:
            w_clusters = getIVPNew(cov_compressed,
                use_extended_terms=use_extended_terms,
                limit_shorts=limit_shorts, b=b_compressed.values)
        w_clusters = pd.Series( w_clusters, index =
            cov_compressed.index )
        # update the weights using the optimal cluster weights
        for clusterId in clustering.keys():
            lbls = clustering[clusterId]
            w[lbls] *= w_clusters[clusterId]
        return w

    return recursive_solve( self.cluster_id )
```

References

[1] López de Prado, Marcos, Building Diversified Portfolios that Outperform Out-of-Sample (May 23, 2016). Journal of Portfolio Management, 2016; https://doi.org/10.3905/jpm.2016.42.4.059. Available at SSRN: https://ssrn.com/abstract=2708678 or http://dx.doi.org/10.2139/ssrn.2708678

[2] López de Prado, Marcos and Lewis, Michael J., Detection of False Investment Strategies Using Unsupervised Learning Methods (August 18, 2018). Available at SSRN: https://ssrn.com/abstract=3167017 or http://dx.doi.org/10.2139/ssrn.3167017

[3] Vim (https://quant.stackexchange.com/users/19004/vim), Maximum Sharpe portfolio (no short selling restrictions) Quantitative Finance Stack Exchange, URL: https://quant.stackexchange.com/questions/43999/maximum-sharpe-portfolio-no-short-selling-restrictions (version: 8/26/2019)

[4] Garey, M. R. and Johnson, D. S., Computers and Intractability. A Guide to the Theory of NP-Completeness (1979).