BUILDING DIVERSIFIED PORTFOLIOS THAT OUTPERFORM OUT-OF-SAMPLE

Marcos López de Prado [†]

First version: December 25, 2015
This version: May 23, 2016

[†] Senior Managing Director, Guggenheim Partners, New York, NY 10017. Research Fellow, Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720. E-mail: lopezdeprado@lbl.gov. Web: www.QuantResearch.org

BUILDING DIVERSIFIED PORTFOLIOS THAT OUTPERFORM OUT-OF-SAMPLE

ABSTRACT

This paper introduces the Hierarchical Risk Parity (HRP) approach. HRP portfolios address three major concerns of quadratic optimizers in general and Markowitz's CLA in particular: Instability, concentration and underperformance.

HRP applies modern mathematics (graph theory and machine learning techniques) to build a diversified portfolio based on the information contained in the covariance matrix. However, unlike quadratic optimizers, HRP does not require the invertibility of the covariance matrix. In fact, HRP can compute a portfolio on an ill-degenerated or even a singular covariance matrix, an impossible feat for quadratic optimizers. Monte Carlo experiments show that HRP delivers lower out-of-sample variance than CLA, even though minimum-variance is CLA's optimization objective. HRP also produces less risky portfolios out-of-sample compared to traditional risk parity methods.

Keywords: Risk parity, tree graph, cluster, dendogram, linkage, metric space.

JEL Classification: G0, G1, G2, G15, G24, E44.
AMS Classification: 91G10, 91G60, 91G70, 62C, 60E.

Portfolio construction is perhaps the most recurrent financial problem. On a daily basis, investment managers must build portfolios that incorporate their views and forecasts on risks and returns. This is the primordial question that a 24 years old Harry Markowitz attempted to answer more than 6 decades ago. His monumental insight was to recognize that various levels of risk are associated with different optimal portfolios in terms of risk-adjusted returns, hence the notion of "efficient frontier" (Markowitz, 1952). One implication was that it is rarely optimal to allocate all assets to the investments with highest expected returns. Instead, we should take into account the correlations across alternative investments in order to build a diversified portfolio.

Before earning his Ph.D. in 1954, Markowitz left academia to work for the RAND Corporation, where he developed the Critical Line Algorithm (CLA). CLA is a quadratic optimization procedure specifically designed for inequality-constrained portfolio optimization problems. This algorithm is notable in that it guarantees that the exact solution is found after a known number of iterations, and that it ingeniously circumvents the Karush-Kuhn-Tucker conditions (Kuhn and Tucker, 1951). A description and open-source implementation of this algorithm can be found in Bailey and López de Prado [2013]. Surprisingly, most financial practitioners still seem unaware of CLA, as they often rely on generic-purpose quadratic programming methods that do not guarantee the correct solution or a stopping time.

Despite of the brilliance of Markowitz's theory, a number of practical problems make CLA solutions somewhat unreliable. A major caveat is that small deviations in the forecasted returns will cause CLA to produce very different portfolios (Michaud, 1998). Given that returns can rarely be forecasted with sufficient accuracy, many authors have opted for dropping them altogether and focus on the covariance matrix. This has led to risk-based asset allocation approaches, of which "risk parity" is a prominent example (Jurczenko, 2015). Dropping the forecasts on returns improves however does not prevent the instability issues. The reason is, quadratic programming methods require the inversion of a positive-definite covariance matrix (all eigenvalues must be positive). This inversion is prone to large errors when the covariance matrix is numerically ill-conditioned, i.e. it has a high condition number (Bailey and López de Prado [2012]).

MARKOWITZ'S CURSE

The condition number of a covariance, correlation (or normal, thus diagonalizable) matrix is the absolute value of the ratio between its maximal and minimal (by moduli) eigenvalues. Figure 1 plots the sorted eigenvalues of several correlation matrices, where the condition number is the ratio between the first and last values of each line. This number is lowest for a diagonal correlation matrix, which is its own inverse. As we add correlated (multicollinear) investments, the condition number grows. At some point, the condition number is so high that numerical errors make the inverse matrix too unstable: A small change on any entry will lead to a very different inverse. This is *Markowitz' curse*: The more correlated the investments, the greater the need for diversification, and yet the more likely we will receive unstable solutions. The benefits of diversification often are more than offset by estimation errors.

[EXHIBIT 1 HERE]

Increasing the size of the covariance matrix will only make matters worse, as each covariance coefficient is estimated with fewer degrees of freedom. In general, we need at least $\frac{1}{2}N(N+1)$ independent and identically distributed (IID) observations in order to estimate a covariance matrix of size N that is not singular. For example, estimating an invertible covariance matrix of size 50 requires at the very least 5 years of daily IID data. As most investors know, correlation structures do not remain invariant over such long periods by any reasonable confidence level. The severity of these challenges is epitomized by the fact that even naïve (equally-weighted) portfolios have been shown to beat mean-variance and risk-based optimization out-of-sample (De Miguel et al., 2009).

FROM GEOMETRIC TO HIERARCHICAL RELATIONSHIPS

These instability concerns have received substantial attention in recent years, as Kolm et al. [2013] have carefully documented. Most alternatives attempt to achieve robustness by incorporating additional constraints (Clarke et al., 2002), introducing Bayesian priors (Black and Litterman, 1992) or improving the numerical stability of the covariance matrix's inverse (Ledoit and Wolf [2003]).

All the methods discussed so far, although published in recent years, are derived from (very) classical areas of mathematics: Geometry, linear algebra and calculus. A correlation matrix is a linear algebra object that measures the cosines of the angles between any two vectors in the vector space formed by the returns series (see Calkin and López de Prado [2014a, 2015b]). One reason for the instability of quadratic optimizers is that the vector space is modelled as a complete (fully connected) graph, where every node is a potential candidate to substitute another. In algorithmic terms, inverting the matrix means evaluating the partial correlations across the complete graph. Exhibit 2a visualizes the relationships implied by a covariance matrix of 50x50, that is 50 nodes and 1225 edges. Small estimation errors are magnified, leading to incorrect solutions. Intuitively it would be desirable to drop unnecessary edges.

[EXHIBITS 2(a)-2(b) HERE]

Let's consider for a moment the practical implications of such topological structure. Suppose that an investor wishes to build a diversified portfolio of securities, including hundreds of stocks, bonds, hedge funds, real estate, private placements, etc. Some investments seem closer substitutes of one another, and other investments seem complementary to one another. For example, stocks could be grouped in terms of liquidity, size, industry and region, where stocks within a given group compete for allocations. In deciding the allocation to a large publicly-traded U.S. financial stock like J.P. Morgan, we will consider adding or reducing the allocation to another large publicly-traded U.S. bank like Goldman Sachs, rather than a small community bank in Switzerland, or a real estate holding in the Caribbean. And yet, to a correlation matrix, all investments are potential substitutes to each other. In other words, correlation matrices lack the notion of *hierarchy*. This lack of hierarchical structure allows weights to vary freely in unintended ways, which is a root cause of CLA's instability. Exhibit 2b visualizes a hierarchical structure known as a tree. A tree structure introduces two desirable features: a) It has only $N-1$ edges to connect N nodes, so the weights only rebalance among peers at various hierarchical levels; and b) the weights are distributed top-down, consistent with how many asset managers build their portfolios, e.g. from asset class to sectors to individual securities. For these reasons, hierarchical structures are designed to give not only stable but also intuitive results.

In this paper we will study a new portfolio construction method that addresses CLA's pitfalls using modern mathematics: Graph theory and machine learning. This Hierarchical Portfolio Construction (HRP) method uses the information contained in the covariance matrix without requiring its inversion or positive-definitiveness. In fact, HRP can compute a portfolio based on a singular covariance matrix, an impossible feat for quadratic optimizers. The algorithm operates in three stages: Tree clustering, quasi-diagonalization and recursive bisection.

STAGE 1: TREE CLUSTERING

Consider a *TxN* matrix of observations *X*, such as returns series of *N* variables over *T* periods. We would like to combine these *N* column-vectors into a hierarchical structure of clusters, so that allocations can flow downstream through a tree graph.

First, we compute a *NxN* correlation matrix with entries $\rho = \{\rho_{i,j}\}_{i,j=1,\dots,N}$, where $\rho_{i,j} = \rho[X_i, X_j]$. We define the distance measure $d: (X_i, X_j) \subset B \rightarrow \mathbb{R} \in [0,1]$, $d_{i,j} = d[X_i, X_j] = \sqrt{\frac{1}{2}(1-\rho_{i,j})}$, where B is the Cartesian product of items in $\{1, \dots, i, \dots, N\}$. This allows us to compute a *NxN* distance matrix $D = \{d_{i,j}\}_{i,j=1,\dots,N}$. Matrix D is a proper metric space (see Appendix A.1 for a proof), in the sense that $d[X,Y] \geq 0$ (non-negativity), $d[X,Y] = 0 \Leftrightarrow X = Y$ (coincidence), $d[X,Y] = d[Y,X]$ (symmetry), and $d[X,Z] \leq d[X,Y] + d[Y,Z]$ (sub-additivity).

$$\{\rho_{i,j}\} = \begin{bmatrix} 1 & .7 & .2 \\ .7 & 1 & -.2 \\ .2 & -.2 & 1 \end{bmatrix} \rightarrow \{d_{i,j}\} = \begin{bmatrix} 0 & .3873 & .6325 \\ .3873 & 0 & .7746 \\ .6325 & .7746 & 0 \end{bmatrix}$$

Example 1 – Encoding a correlation matrix ρ as a distance matrix D

Second, we compute the Euclidean distance between any two column-vectors of *D*, $\tilde{d}: (D_i, D_j) \subset B \rightarrow \mathbb{R} \in [0, \sqrt{N}]$, $\tilde{d}_{i,j} = \tilde{d}[D_i, D_j] = \sqrt{\sum_{n=1}^{N} (d_{n,i} - d_{n,j})^2}$. Note the difference between distance metrics $d_{i,j}$ and $\tilde{d}_{i,j}$. Whereas $d_{i,j}$ is defined on column-vectors of *X*, $\tilde{d}_{i,j}$ is defined on column-vectors of *D* (a distance of distances). Therefore, $\tilde{d}$ is a distance defined over the entire metric space *D*, as each $\tilde{d}_{i,j}$ is a function of the entire correlation matrix (rather than a particular cross-correlation pair).

$$\{d_{i,j}\} = \begin{bmatrix} 0 & .3873 & .6325 \\ .3873 & 0 & .7746 \\ .6325 & .7746 & 0 \end{bmatrix} \rightarrow \{\tilde{d}_{i,j}\}_{i,j=\{1,2,3\}}$$
$$= \begin{bmatrix} 0 & .5659 & .9747 \\ .5659 & 0 & 1.1225 \\ .9747 & 1.1225 & 0 \end{bmatrix}$$

Example 2 – Euclidean distance of correlation distances

Third, we cluster together the pair of columns (i^*, j^*) such that $(i^*, j^*) = \operatorname{argmin}_{\substack{(i,j)\\ i\neq j}}\{\tilde{d}_{i,j}\}$, and denote this cluster as $u[1]$.

$$\{\tilde{d}_{i,j}\}_{i,j=\{1,2,3\}} = \begin{bmatrix} 0 & .5659 & .9747 \\ .5659 & 0 & 1.1225 \\ .9747 & 1.1225 & 0 \end{bmatrix} \rightarrow u[1] = (1,2)$$

Example 3 – Clustering items

Fourth, we need to define the distance between a newly formed cluster $u[1]$ and the single (unclustered) items, so that $\{\tilde{d}_{i,j}\}$ may be updated. In hierarchical clustering analysis, this is known as the "linkage criterion". For example, we can define the distance between an item i of $\tilde{d}$ and the new cluster $u[1]$ as $\dot{d}_{i,u[1]} = \min\left[\{\tilde{d}_{i,j}\}_{j\in u[1]}\right]$ (the nearest point algorithm).

$$u[1] = (1,2) \rightarrow \{\dot{d}_{i,u[1]}\} = \begin{bmatrix} \min[0, .5659] \\ \min[.5659, 0] \\ \min[.9747, 1.1225] \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ .9747 \end{bmatrix}$$

Example 4 – Updating matrix $\{\tilde{d}_{i,j}\}$ with the new cluster u

Fifth, matrix $\{\tilde{d}_{i,j}\}$ is updated by appending $\dot{d}_{i,u[1]}$ and dropping the clustered columns and rows $j \in u[1]$.

$$\{\tilde{d}_{i,j}\}_{i,j=\{1,2,3,4\}} = \begin{bmatrix} 0 & .5659 & .9747 & 0 \\ .5659 & 0 & 1.1225 & 0 \\ .9747 & 1.1225 & 0 & .9747 \\ 0 & 0 & .9747 & 0 \end{bmatrix}$$

$$\{\tilde{d}_{i,j}\}_{i,j=\{3,4\}} = \begin{bmatrix} 0 & .9747 \\ .9747 & 0 \end{bmatrix}$$

Example 5 – Updating matrix $\{\tilde{d}_{i,j}\}$ with the new cluster u

Sixth, applied recursively, steps 3-5 allow us to append $N-1$ such clusters to matrix D, at which point the final cluster contains all of the original items and the clustering algorithm stops.

$$\{\tilde{d}_{i,j}\}_{i,j=\{3,4\}} = \begin{bmatrix} 0 & .9747 \\ .9747 & 0 \end{bmatrix} \rightarrow u[2] = (3,4) \rightarrow \text{Stop}$$

Example 6 – Recursion in search of remaining clusters

Exhibit 3 displays the clusters formed at each iteration for this example, as well as the distances $\tilde{d}_{i^*,j^*}$ that triggered every cluster (step 3). This procedure can be applied to a wide array of distance metrics $d_{i,j}$, $\tilde{d}_{i,j}$ and $\dot{d}_{i,u}$, beyond those illustrated in this paper. See Rokach and Maimon [2005] for alternative metrics, the discussion on Fiedler's vector and Stewart's spectral clustering method in Brualdi [2010], as well as algorithms in the scipy library.[1]

[1] For additional metrics see:

[EXHIBIT 3 HERE]

This stage allows us to define a linkage matrix as a $(N-1)x4$ matrix with structure $Y = \{(y_{m,1}, y_{m,2}, y_{m,3}, y_{m,4})\}_{m=1,\ldots,N-1}$, i.e. with one 4-tuple per cluster. Items $(y_{m,1}, y_{m,2})$ report the constituents. Item $y_{m,3}$ reports the distance between $y_{m,1}$ and $y_{m,2}$, that is $y_{m,3} = \tilde{d}_{y_{m,1},y_{m,2}}$. Item $y_{m,3} \leq N$ reports the number of original items included in cluster m.

STAGE 2: QUASI-DIAGONALIZATION

This stage reorganizes the rows and columns of the covariance matrix, so that the largest values lie along the diagonal. This quasi-diagonalization of the covariance matrix (without requiring a change of basis) renders a useful property: Similar investments are placed together, and dissimilar investments are placed far apart (see Exhibits 5-6 for an example). The algorithm works as follows: We know that each row of the linkage matrix merges two branches into one. We replace clusters in $(y_{N-1,1}, y_{N-1,2})$ with their constituents recursively, until no clusters remain. These replacements preserve the order of the clustering. The output is a sorted list of original (unclustered) items. This is implemented in Code snippet 1.

```
#------------------------------------------------------------------------------
def getQuasiDiag(link):
    # Sort clustered items by distance
    link=link.astype(int)
    sortIx=pd.Series([link[-1,0],link[-1,1]])
    numItems=link[-1,3] # number of original items
    while sortIx.max()>=numItems:
        sortIx.index=range(0,sortIx.shape[0]*2,2) # make space
        df0=sortIx[sortIx>=numItems] # find clusters
        i=df0.index;j=df0.values-numItems
        sortIx[i]=link[j,0] # item 1
        df0=pd.Series(link[j,1],index=i+1)
        sortIx=sortIx.append(df0) # item 2
        sortIx=sortIx.sort_index() # re-sort
        sortIx.index=range(sortIx.shape[0]) # re-index
    return sortIx.tolist()
```

Code snippet 1 – Quasi-diagonalization

STAGE 3: RECURSIVE BISECTION

Stage 2 has delivered a quasi-diagonal matrix. The inverse-variance allocation is optimal for a diagonal covariance matrix (see Appendix A.2 for a proof). We can take advantage of these facts in two different ways: a) Bottom-up, to define the variance of a continuous subset as the variance

---

http://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html
http://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.cluster.hierarchy.linkage.html

of an inverse-variance allocation; b) top-down, to split allocations between adjacent subsets in inverse proportion to their aggregated variances. The following algorithm formalizes this idea:

1. The algorithm is initialized by:
   a. setting the list of items: $L = \{L_0\}$, with $L_0 = \{n\}_{n=1,\ldots,N}$
   b. assigning a unit weight to all items: $w_n = 1, \forall n = 1, \ldots, N$
2. If $|L_i| = 1, \forall L_i \in L$, then stop
3. For each $L_i \in L$ such that $|L_i| > 1$:
   a. bisect L_i into two subsets, $L_i^{(1)} \cup L_i^{(2)} = L_i$, where $\left|L_i^{(1)}\right| = \text{int}\left[\frac{1}{2}|L_i|\right]$, and the order is preserved
   b. define the variance of $L_i^{(j)}, j = 1,2$, as the quadratic form $\tilde{V}_i^{(j)} \equiv \tilde{w}_i^{(j)\prime} V_i^{(j)} \tilde{w}_i^{(j)}$, where $V_i^{(j)}$ is the covariance matrix between the constituents of the $L_i^{(j)}$ bisection, and $\tilde{w}_i^{(j)} = \text{diag}\left[V_i^{(j)}\right]^{-1} \frac{1}{\text{tr}\left[\text{diag}\left[V_i^{(j)}\right]^{-1}\right]}$, where $\text{diag}[.]$ and $\text{tr}[.]$ are the diagonal and trace operators
   c. compute the split factor: $\alpha_i = 1 - \frac{\tilde{V}_i^{(1)}}{\tilde{V}_i^{(1)} + \tilde{V}_i^{(2)}}$, so that $0 \leq \alpha_i \leq 1$
   d. re-scale allocations w_n by a factor of α_i, $\forall n \in L_i^{(1)}$
   e. re-scale allocations w_n by a factor of $(1 - \alpha_i)$, $\forall n \in L_i^{(2)}$
4. Loop to step 2

Step 3.b takes advantage of the quasi-diagonalization bottom-up, because it defines the variance of the partition $L_i^{(j)}$ using inverse-variance weightings $\tilde{w}_i^{(j)}$. *Step 3.c* takes advantage of the quasi-diagonalization top-down, because it splits the weight in inverse proportion to the cluster's variance. This algorithm guarantees that $0 \leq w_i \leq 1$, $\forall i = 1, \ldots, N$, and $\sum_{i=1}^{N} w_i = 1$, because at each iteration we are splitting the weights received from higher hierarchical levels. Constraints can be easily introduced in this stage, by replacing the equations in steps 3.c-3.e according to the user's preferences. Stage 3 is implemented in Code Snippet 2.

```
#------------------------------------------------------------------------------
def getRecBipart(cov,sortIx):
    # Compute HRP alloc
    w=pd.Series(1,index=sortIx)
    cItems=[sortIx] # initialize all items in one cluster
    while len(cItems)>0:
        cItems=[i[j:k] for i in cItems for j,k in ((0,len(i)/2), \
            (len(i)/2,len(i))) if len(i)>1] # bi-section
        for i in xrange(0,len(cItems),2): # parse in pairs
            cItems0=cItems[i] # cluster 1
            cItems1=cItems[i+1] # cluster 2
            cVar0=getClusterVar(cov,cItems0)
            cVar1=getClusterVar(cov,cItems1)
            alpha=1-cVar0/(cVar0+cVar1)
```

```
        w[cItems0]*=alpha # weight 1
        w[cItems1]*=1-alpha # weight 2
    return w
```

Code snippet 2 – Recursive bisection

This concludes a first description of the HRP algorithm, which solves the allocation problem in deterministic logarithmic time, $T(n) = O(\log_2 n)$. Next, we will put to practice what we have learned, and evaluate the method's accuracy out of sample.

A NUMERICAL EXAMPLE

We begin by simulating a matrix of observations X, of order $(10000x10)$. The correlation matrix is visualized in Exhibit 4 as a heatmap. Exhibit 5 displays the dendogram of the resulting clusters (stage 1). Exhibit 6 shows the same correlation matrix, reorganized in blocks according to the identified clusters (stage 2). Appendix A.3 provides the code used to generate this numerical example.

[EXHIBITS 4-6 HERE]

On this random data, we compute HRP's allocations (stage 3), and compare them to the allocations from two competing methodologies: 1) Quadratic optimization, as represented by CLA's minimum-variance portfolio (the only portfolio of the efficient frontier that does not depend on returns' means); and 2) traditional risk parity, exemplified by the Inverse-Variance Portfolio (IVP). See Bailey and López de Prado [2013] for a comprehensive implementation of CLA, and Appendix A.2 for a derivation of IVP. We apply the standard constraints that $0 \leq w_i \leq 1$ (non-negativity), $\forall i = 1, \ldots, N$, and $\sum_{i=1}^{N} w_i = 1$ (full investment). Incidentally, the condition number for the covariance matrix in this example is only 150.9324, not particularly high and therefore not unfavorable to CLA.

[EXHIBIT 7 HERE]

From the allocations in Exhibit 7, we can appreciate a few stylized features: First, CLA concentrates 92.66% of the allocation on the top 5 holdings, while HRP concentrates only 62.57%. Second, CLA assigns zero weight to 3 investments (without the $0 \leq w_i$ constraint, the allocation would have been negative). Third, HRP seems to find a compromise between CLA's concentrated solution and traditional risk parity's IVP allocation. The reader can use the code in Appendix A.3 to verify that these findings generally hold for alternative random covariance matrices.

What drives CLA's extreme concentration is its goal of minimizing the portfolio's risk. And yet both portfolios have a very similar standard deviation ($\sigma_{HRP} = 0.4640$, $\sigma_{CLA} = 0.4486$). So CLA has discarded half of the investment universe in favor of a minor risk reduction. The reality of course is, CLA's portfolio is deceitfully diversified, because any distress situation affecting the five top allocations will have a much greater negative impact on CLA's than HRP's portfolio.

OUT-OF-SAMPLE MONTE CARLO SIMULATIONS

In our numerical example, CLA's portfolio has lower risk than HRP's *in-sample*. However, the portfolio with minimum variance in-sample is not necessarily the one with minimum variance *out-of-sample*. It would be all too easy for us to pick a particular historical dataset where HRP outperforms CLA and IVP (for a discussion on overfitting and selection bias, see Bailey and López de Prado [2015]). Instead, in this section we evaluate via Monte Carlo the performance out-of-sample of HRP against CLA's minimum-variance and traditional risk parity's IVP allocations. This will also help us understand what features make a method preferable to the rest, regardless of anecdotal counter-examples.

First, we generate 10 series of random Gaussian returns (520 observations, equivalent to 2 years of daily history), with 0 mean and an arbitrary standard deviation of 10%. Real prices exhibit frequent jumps (Merton, 1976) and returns are not cross-sectionally independent, so we must add random shocks and a random correlation structure to our generated data. Second, we compute HRP, CLA and IVP portfolios by looking back at 260 observations (a year of daily history). These portfolios are re-estimated and rebalanced every 22 observations (equivalent to a monthly frequency). Third, we compute the out-of-sample returns associated with those three portfolios. This procedure is repeated 10,000 times.

All mean portfolio returns out-of-sample are essentially 0, as expected. The critical difference comes from the variance of the out-of-sample portfolio returns: $\sigma^2_{CLA} = 0.1157$, $\sigma^2_{IVP} = 0.0928$ and $\sigma^2_{HRP} = 0.0671$. Although CLA's goal is to deliver the lowest variance (that is the objective of its optimization program), its performance happens to exhibit the highest variance out-of-sample, and 72.47% greater variance than HRP's. In other words, HRP would improve the out-of-sample Sharpe ratio of a CLA strategy by about 31.3%, a rather significant boost. Assuming that the covariance matrix is diagonal brings some stability to the IVP, however its variance is still 38.24% greater than HRP's. This variance reduction out-of-sample is critically important to risk parity investors, given their use of substantial leverage. See Bailey et al. [2014] for a broader discussion of in-sample vs. out-of-sample performance.

The mathematical proof for HRP's outperformance over Markowitz's CLA and traditional risk parity's IVP is somewhat involved and beyond the scope of this introductory paper. In intuitive terms, we can understand the above empirical results as follows: Shocks affecting a specific investment penalize CLA's concentration. Shocks involving several correlated investments penalize IVP's ignorance of the correlation structure. HRP provides better protection against both, common and idiosyncratic shocks, by finding a compromise between diversification across all investments and diversification across clusters of investments at multiple hierarchical levels. Exhibit 8 plots the time series of allocations for the first of the 10,000 runs.

[EXHIBITS 8(a)-8(c) HERE]

Appendix A.4 provides the python code that implements the above study. The reader can experiment with different parameter configurations and reach similar conclusions. In particular, HRP's out-of-sample outperformance becomes even more substantial for larger investment universes, or when more shocks are added or a stronger correlation structure is considered, or rebalancing costs are taken into account.

FURTHER RESEARCH

The methodology introduced in this paper is flexible, scalable and admits multiple variations of the same ideas. Using the code provided, readers can research and evaluate what HRP configurations work best for their particular problem. For example, at stage 1 they can apply alternative definitions of $d_{i,j}$, $\tilde{d}_{i,j}$ and $\dot{d}_{i,u}$, or clustering algorithms; at stage 3, they can use different functions for $\widetilde{w}_m$ and α, or alternative allocation constraints. Instead of carrying out a recursive bisection, stage 3 could also split allocations top-down using the clusters from stage 1.

In a future note, we will show that it is relatively straightforward to incorporate forecasted returns and Black-Litterman-style views to this hierarchical approach. In fact, the inquisitive reader may have realized that, at its core, HRP is essentially a robust procedure to avoid matrix inversions, and the same ideas underlying HRP can be used to replace many econometric regression methods, notorious for their unstable outputs (like VAR or VECM). Exhibit 9 displays a large correlation matrix of fixed income securities before and after clustering, with over 2.1 million entries. Traditional optimization or econometric methods fail to recognize the hierarchical structure of financial Big Data, where the numerical instabilities defeat the benefits of the analysis, resulting in unreliable and detrimental outcomes.

[EXHIBIT 9 HERE]

CONCLUSIONS

Although mathematically correct, quadratic optimizers in general, and Markowitz's CLA in particular, are known to deliver generally unreliable solutions due to their instability, concentration and underperformance. The root cause for these issues is that quadratic optimizers require the inversion of a covariance matrix. Markowitz's curse is that the more correlated investments are, the greater is the need for a diversified portfolio, and yet the greater are that portfolio's estimation errors.

In this paper we have exposed a major source of quadratic optimizers' instability: A matrix of size N is associated with a complete graph with $\frac{1}{2}N(N-1)$ edges. With so many edges connecting the nodes of the graph, weights are allowed to rebalance with complete freedom. This lack of hierarchical structure means that small estimation errors will lead to entirely different solutions. HRP replaces the covariance structure with a tree structure, accomplishing three goals: a) Unlike traditional risk parity methods, it fully utilizes the information contained in the covariance matrix, b) weights' stability is recovered and c) the solution is intuitive by construction. The algorithm converges in deterministic logarithmic time.

HRP is robust, visual and flexible, allowing the user to introduce constraints or manipulate the tree structure without compromising the algorithm's search. These properties are derived from the fact that HRP does not require covariance invertibility. Indeed, HRP can compute a portfolio on an ill-degenerated or even a singular covariance matrix, an impossible feat for quadratic optimizers.

This note focuses on a portfolio construction application, however the reader will find other practical uses for making decisions under uncertainty, particularly in the presence of a nearly-singular covariance matrix: Capital allocation to portfolio managers, allocations across algorithmic strategies, bagging and boosting of machine learning signals, forecasts from random forests, replacement to unstable econometric models (VAR, VECM), etc.

Of course, quadratic optimizers like CLA produce the minimum-variance portfolio in-sample (that is its objective function). Monte Carlo experiments show that HRP delivers lower out-of-sample variance than CLA or traditional risk parity methods (IVP). Since Bridgewater pioneered risk parity in the 1990s, some of the largest asset managers have launched funds that follow this approach, for combined assets in excess of $500 billion. Given their extensive use of leverage, these funds should benefit from adopting a more stable risk parity allocation method, thus achieving superior risk-adjusted returns and lower rebalance costs.

APPENDIX

A.1. CORRELATION-BASED METRIC

Consider two real-valued vectors X, Y of size T, and a correlation variable $\rho[X,Y]$, with the only requirement that $\sigma[X,Y]=\rho[X,Y]\sigma[X]\sigma[Y]$, where $\sigma[X,Y]$ is the covariance between the two vectors, and $\sigma[.]$ is the standard deviation. Note that Pearson's is not the only correlation to satisfy these requirements.

Let's prove that $d[X,Y]=\sqrt{\frac{1}{2}(1-\rho[X,Y])}$ is a true metric. First, the Euclidean distance between the two vectors is $d[X,Y]=\sqrt{\sum_{t=1}^{T}(X_t-Y_t)^2}$. Second, we z-standardize those vectors as $x=\frac{X-\bar{X}}{\sigma[X]}$, $y=\frac{Y-\bar{Y}}{\sigma[Y]}$. Consequently, $0\leq\rho[x,y]=\rho[X,Y]$. Third, we derive the Euclidean distance $d[x,y]$ as,

$$d[x,y]=\sqrt{\sum_{t=1}^{T}(x_t-y_t)^2}=\sqrt{\sum_{t=1}^{T}x_t^2+\sum_{t=1}^{T}y_t^2-2\sum_{t=1}^{T}x_t y_t}=\sqrt{T+T-2T\sigma[x,y]}$$
$$=\sqrt{2T\left(1-\underbrace{\rho[x,y]}_{=\rho[X,Y]}\right)}=\sqrt{4T}d[X,Y]$$

In other words, the distance $d[X,Y]$ is a linear multiple of the Euclidean distance between the vectors $\{X,Y\}$ after z-standardization, hence it inherits the true-metric properties of the Euclidean distance.

Similarly, we can prove that $d[X,Y]=\sqrt{1-|\rho[X,Y]|}$ is also a true metric. In order to do that, we redefine $y=\frac{Y-\bar{Y}}{\sigma[Y]}\operatorname{sgn}[\rho[X,Y]]$, where $\operatorname{sgn}[.]$ is the sign operator, so that $0\leq\rho[x,y]=|\rho[X,Y]|$. Then,

$$d[x,y]=\sqrt{2T\left(1-\underbrace{\rho[x,y]}_{=|\rho[X,Y]|}\right)}=\sqrt{2T}d[X,Y]$$

A.2. INVERSE VARIANCE ALLOCATION

Stage 3 splits a weight in inverse proportion to the subset's variance. We can proof that such allocation is optimal when the covariance matrix is diagonal. Consider the standard quadratic optimization problem of size N,

$$\min_{\omega} \omega' V \omega$$
$$\text{s.t.}: \omega' a = 1_I$$

with solution $\omega = \frac{V^{-1}a}{a'V^{-1}a}$. For the characteristic vector $a = 1_N$, the solution is the minimum variance portfolio. If V is diagonal, $\omega_n = \frac{V_{n,n}^{-1}}{\sum_{i=1}^{N} V_{i,i}^{-1}}$. In the particular case of $N = 2$, $\omega_1 = \frac{\frac{1}{V_{1,1}}}{\frac{1}{V_{1,1}}+\frac{1}{V_{2,2}}} = 1 - \frac{V_{1,1}}{V_{1,1}+V_{2,2}}$, which is how stage 3 splits a weight between two bisections of a subset.

A.3. REPRODUCING THE NUMERICAL EXAMPLE

The following code can be used to reproduce our results and simulate additional numerical examples. Function **generateData()** produces a matrix of time series where a number *size0* of vectors are uncorrelated, and a number *size1* of vectors are correlated. The reader can change the *np.random.seed* in **generateData()** to run alternative examples and gain an intuition of how HRP works. Scipy's function **linkage()** can be used to perform stage 1, function **getQuasiDiag()** performs stage 2, and function **getRecBipart()** carries out stage 3.

```
# On 20151227 by MLdP <lopezdeprado@lbl.gov>
# Hierarchical Risk Parity
import matplotlib.pyplot as mpl
import scipy.cluster.hierarchy as sch,random,numpy as np,pandas as pd
#------------------------------------------------------------------------------
def getIVP(cov,**kargs):
    # Compute the inverse-variance portfolio
    ivp=1./np.diag(cov)
    ivp/=ivp.sum()
    return ivp
#------------------------------------------------------------------------------
def getClusterVar(cov,cItems):
    # Compute variance per cluster
    cov_=cov.loc[cItems,cItems] # matrix slice
    w_=getIVP(cov_).reshape(-1,1)
    cVar=np.dot(np.dot(w_.T,cov_),w_)[0,0]
    return cVar
#------------------------------------------------------------------------------
def getQuasiDiag(link):
    # Sort clustered items by distance
    link=link.astype(int)
    sortIx=pd.Series([link[-1,0],link[-1,1]])
    numItems=link[-1,3] # number of original items
    while sortIx.max()>=numItems:
        sortIx.index=range(0,sortIx.shape[0]*2,2) # make space
        df0=sortIx[sortIx>=numItems] # find clusters
```

```
        i=df0.index;j=df0.values-numItems
        sortIx[i]=link[j,0] # item 1
        df0=pd.Series(link[j,1],index=i+1)
        sortIx=sortIx.append(df0) # item 2
        sortIx=sortIx.sort_index() # re-sort
        sortIx.index=range(sortIx.shape[0]) # re-index
    return sortIx.tolist()
#------------------------------------------------------------------------------
def getRecBipart(cov,sortIx):
    # Compute HRP alloc
    w=pd.Series(1,index=sortIx)
    cItems=[sortIx] # initialize all items in one cluster
    while len(cItems)>0:
        cItems=[i[j:k] for i in cItems for j,k in ((0,len(i)/2), \
            (len(i)/2,len(i))) if len(i)>1] # bi-section
        for i in xrange(0,len(cItems),2): # parse in pairs
            cItems0=cItems[i] # cluster 1
            cItems1=cItems[i+1] # cluster 2
            cVar0=getClusterVar(cov,cItems0)
            cVar1=getClusterVar(cov,cItems1)
            alpha=1-cVar0/(cVar0+cVar1)
            w[cItems0]*=alpha # weight 1
            w[cItems1]*=1-alpha # weight 2
    return w
#------------------------------------------------------------------------------
def correlDist(corr):
    # A distance matrix based on correlation, where 0<=d[i,j]<=1
    # This is a proper distance metric
    dist=((1-corr)/2.)**.5 # distance matrix
    return dist
#------------------------------------------------------------------------------
def plotCorrMatrix(path,corr,labels=None):
    # Heatmap of the correlation matrix
    if labels is None:labels=[]
    mpl.pcolor(corr)
    mpl.colorbar()
    mpl.yticks(np.arange(.5,corr.shape[0]+.5),labels)
    mpl.xticks(np.arange(.5,corr.shape[0]+.5),labels)
    mpl.savefig(path)
    mpl.clf();mpl.close() # reset pylab
    return
#------------------------------------------------------------------------------
def generateData(nObs,size0,size1,sigma1):
    # Time series of correlated variables
    #1) generating some uncorrelated data
    np.random.seed(seed=12345);random.seed(12345)
```

```
    x=np.random.normal(0,1,size=(nObs,size0)) # each row is a variable
    #2) creating correlation between the variables
    cols=[random.randint(0,size0-1) for i in xrange(size1)]
    y=x[:,cols]+np.random.normal(0,sigma1,size=(nObs,len(cols)))
    x=np.append(x,y,axis=1)
    x=pd.DataFrame(x,columns=range(1,x.shape[1]+1))
    return x,cols
#------------------------------------------------------------------------------
def main():
    #1) Generate correlated data
    nObs,size0,size1,sigma1=10000,5,5,.25
    x,cols=generateData(nObs,size0,size1,sigma1)
    print [(j+1,size0+i) for i,j in enumerate(cols,1)]
    #2) compute and plot correl matrix
    cov,corr=x.cov(),x.corr()
    plotCorrMatrix('HRP3_corr0.png',corr,labels=corr.columns)
    #3) cluster
    dist=correlDist(corr)
    link=sch.linkage(dist,'single')
    sortIx=getQuasiDiag(link)
    sortIx=corr.index[sortIx].tolist() # recover labels
    df0=corr.loc[sortIx,sortIx] # reorder
    plotCorrMatrix('HRP3_corr1.png',df0,labels=df0.columns)
    #4) Capital allocation
    hrp=getRecBipart(cov,sortIx)
    print hrp
    return
#------------------------------------------------------------------------------
if __name__=='__main__':main()
```

A.4. REPRODUCING THE MONTE CARLO EXPERIMENT

The following code implements Monte Carlo experiments on three allocation methods: HRP, CLA and IVP. All libraries are standard except for HRP, which is provided in Appendix A.2, and CLA, which can be found in Bailey and López de Prado [2013]. The subroutine **generateData()** simulates the correlated data, with two types of random shocks: Common to various investments and specific to a single investment. There are two shocks of each type, one positive and one negative. The variables for the experiments are set as arguments of **hrpMC()**. They were chosen arbitrarily, and the user can experiment with alternative combinations.

```
# On 20151231 by MLdP <lopezdeprado@lbl.gov>
import scipy.cluster.hierarchy as sch,random,numpy as np,pandas as pd,CLA
from HRP import correlDist,getIVP,getQuasiDiag,getRecBipart
#------------------------------------------------------------------------------
def generateData(nObs,sLength,size0,size1,mu0,sigma0,sigma1F):
    # Time series of correlated variables
```

```
    #1) generate random uncorrelated data
    x=np.random.normal(mu0,sigma0,size=(nObs,size0)) # each row is a variable
    #2) create correlation between the variables
    cols=[random.randint(0,size0-1) for i in xrange(size1)]
    y=x[:,cols]+np.random.normal(0,sigma0*sigma1F,size=(nObs,len(cols)))
    x=np.append(x,y,axis=1)
    #3) add common random shock
    point=np.random.randint(sLength,nObs-1,size=2)
    x[np.ix_(point,[cols[0],size0])]=np.array([[-.5,-.5],[2,2]])
    #4) add specific random shock
    point=np.random.randint(sLength,nObs-1,size=2)
    x[point,cols[-1]]=np.array([-.5,2])
    return x,cols
#------------------------------------------------------------------------------
def getHRP(cov,corr):
    # Construct a hierarchical portfolio
    corr,cov=pd.DataFrame(corr),pd.DataFrame(cov)
    dist=correlDist(corr)
    link=sch.linkage(dist,'single')
    sortIx=getQuasiDiag(link)
    sortIx=corr.index[sortIx].tolist() # recover labels
    hrp=getRecBipart(cov,sortIx)
    return hrp.sort_index()
#------------------------------------------------------------------------------
def getCLA(cov,**kargs):
    # Compute CLA's minimum variance portfolio
    mean=np.arange(cov.shape[0]).reshape(-1,1) # Not used by C portf
    lB=np.zeros(mean.shape)
    uB=np.ones(mean.shape)
    cla=CLA.CLA(mean,cov,lB,uB)
    cla.solve()
    return cla.w[-1].flatten()
#------------------------------------------------------------------------------
def hrpMC(numIters=1e4,nObs=520,size0=5,size1=5,mu0=0,sigma0=1e-2, \
    sigma1F=.25,sLength=260,rebal=22):
    # Monte Carlo experiment on HRP
    methods=[getIVP,getHRP,getCLA]
    stats,numIter={i.__name__:pd.Series() for i in methods},0
    pointers=range(sLength,nObs,rebal)
    while numIter<numIters:
        print numIter
        #1) Prepare data for one experiment
        x,cols=generateData(nObs,sLength,size0,size1,mu0,sigma0,sigma1F)
        r={i.__name__:pd.Series() for i in methods}
        #2) Compute portfolios in-sample
        for pointer in pointers:
```

```
            x_=x[pointer-sLength:pointer]
            cov_,corr_=np.cov(x_,rowvar=0),np.corrcoef(x_,rowvar=0)
            #3) Compute performance out-of-sample
            x_=x[pointer:pointer+rebal]
            for func in methods:
                w_=func(cov=cov_,corr=corr_) # callback
                r_=pd.Series(np.dot(x_,w_))
                r[func.__name__]=r[func.__name__].append(r_)
        #4) Evaluate and store results
        for func in methods:
            r_=r[func.__name__].reset_index(drop=True)
            p_=(1+r_).cumprod()
            stats[func.__name__].loc[numIter]=p_.iloc[-1]-1 # terminal return
        numIter+=1
    #5) Report results
    stats=pd.DataFrame.from_dict(stats,orient='columns')
    stats.to_csv('stats.csv')
    df0,df1=stats.std(),stats.var()
    print pd.concat([df0,df1,df1/df1['getHRP']-1],axis=1)
    return
#------------------------------------------------------------------------------
if __name__=='__main__':hrpMC()
```

EXHIBITS

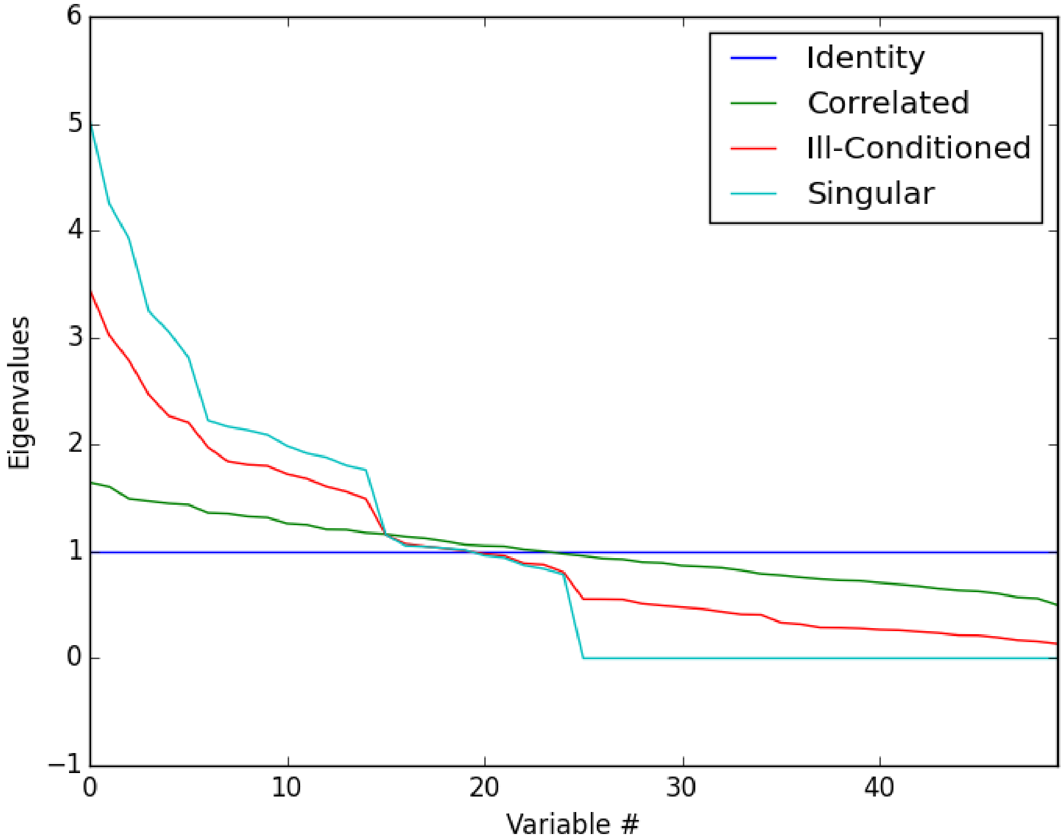

Exhibit 1 – Visualization of Markowitz's curse

A diagonal correlation matrix has the lowest condition number. As we add correlated investments, the maximum eigenvalue is greater and the minimum eigenvalue is lower. The condition number rises quickly, leading to unstable inverse correlation matrices. At some point, the benefits of diversification are more than offset by estimation errors.

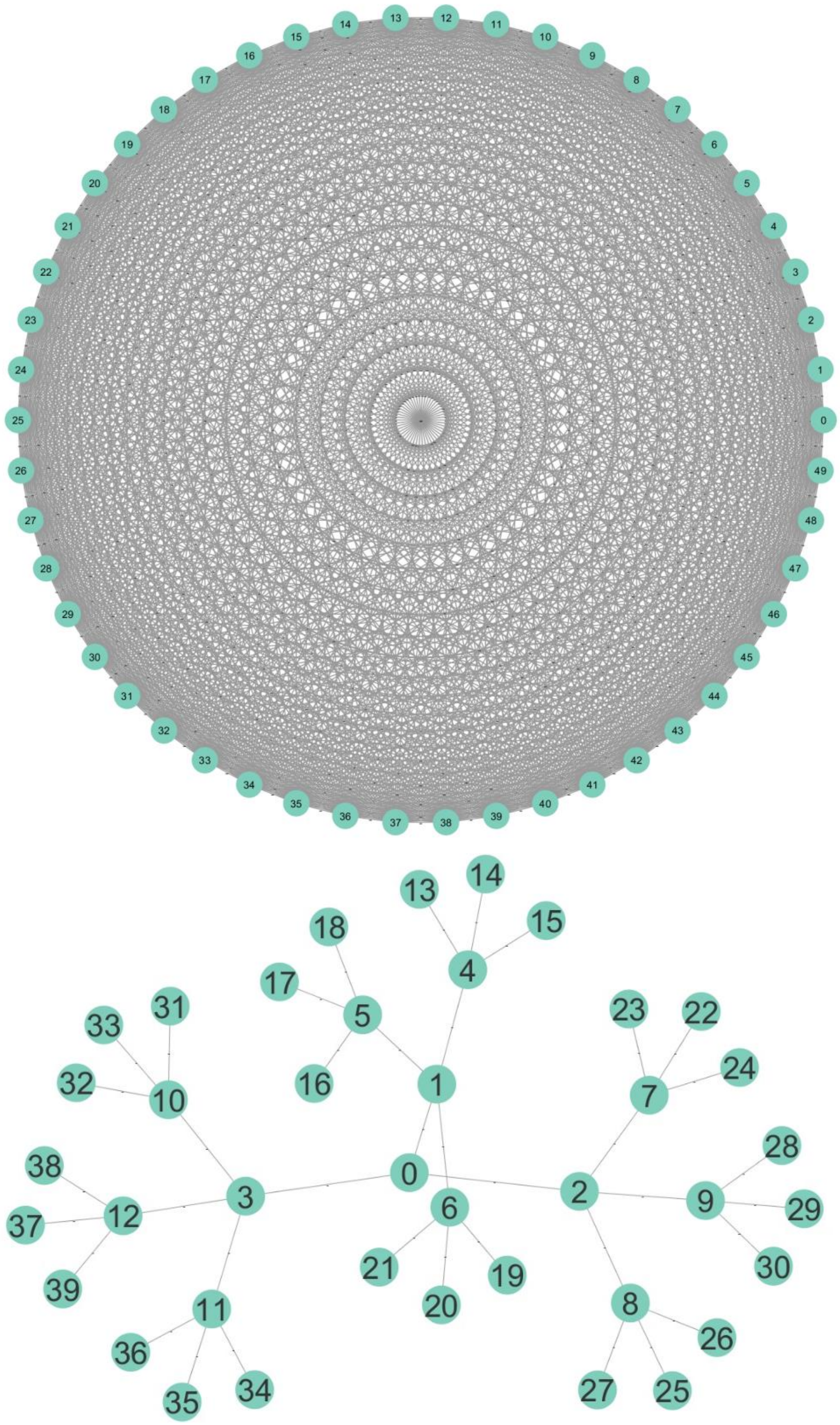

Exhibit 2 – The complete-graph (top) and the tree-graph (bottom) structures

Correlation matrices can be represented as complete graphs, which lack the notion of hierarchy: Each investment is substitutable with another. In contrast, tree structures incorporate hierarchical relationships.

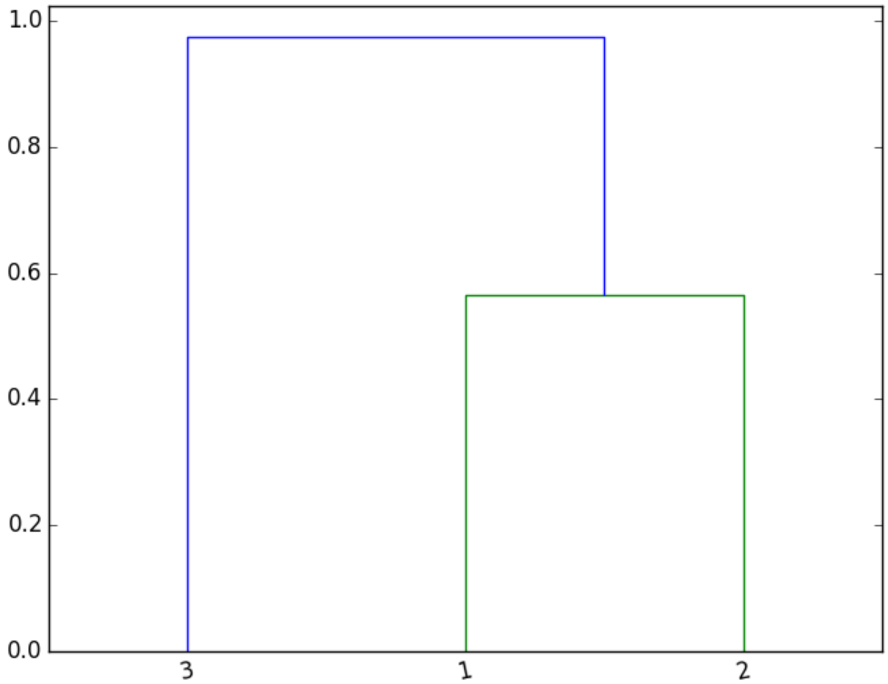

Exhibit 3 – Sequence of cluster formation

A tree structure derived from our numerical example, here plotted as a dendogram. The y-axis measures the distance between the two merging leaves.

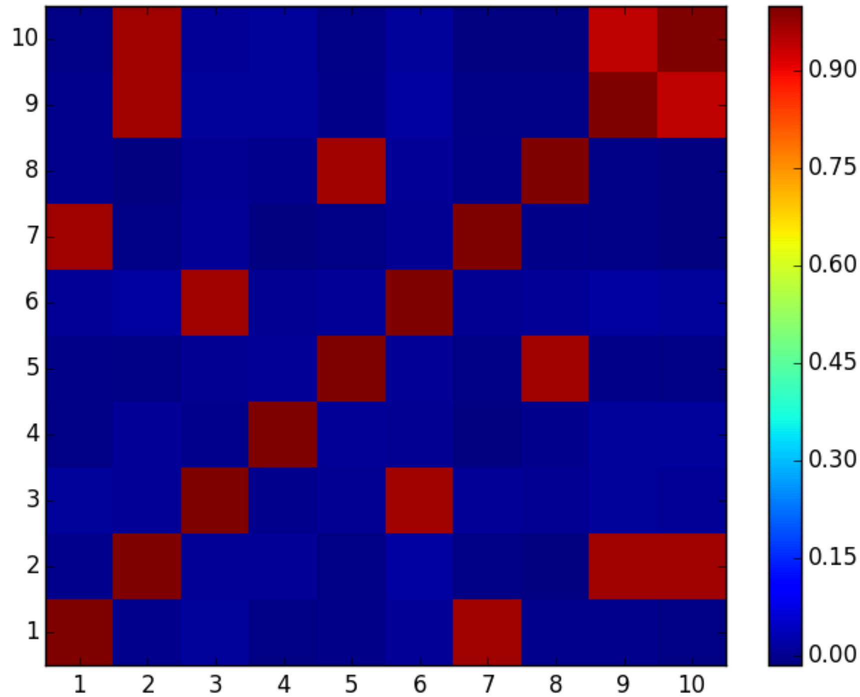

Exhibit 4 – Heatmap of original covariance matrix

This correlation matrix has been computed on random series $X = \{X_i\}_{i=1,\dots,10}$ drawn as follows. First, we draw 5 random vectors from a standard Normal distribution, $\{X_j = z\}_{j=1,\dots,5}$. Second, we draw 5 random integer numbers from a uniform distribution, with replacement, $\vartheta = \{\vartheta_k\}_{k=1,\dots,5}$. Third, we compute $X_{5+k} = X_{\vartheta_k} + \frac{1}{4}z, \forall k = 1, \dots, 5$. This forces the 5 last columns to be partially correlated to some of the first 5 series.

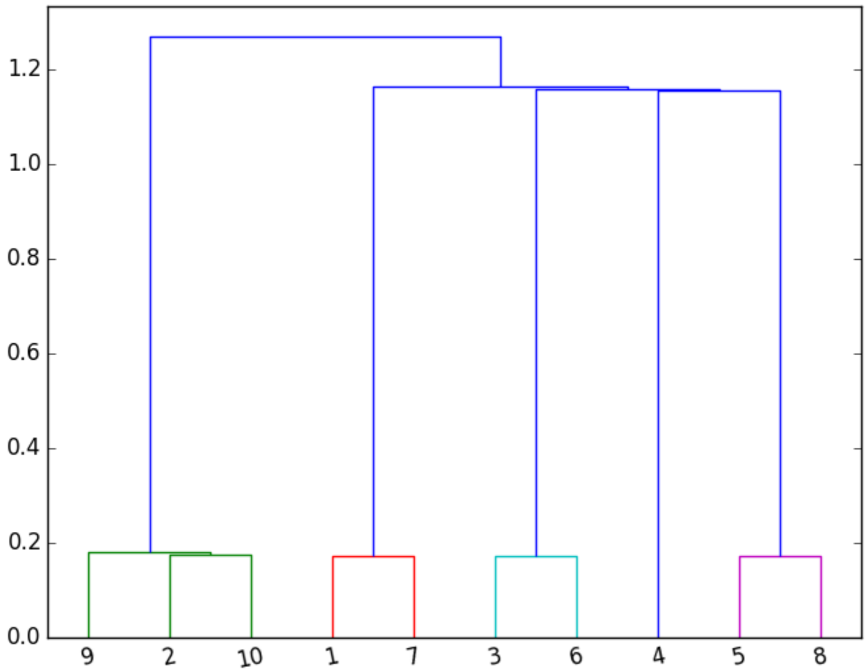

Exhibit 5 – Dendogram of cluster formation

The clustering procedure has correctly identified that series 9 and 10 where perturbations of series 2, hence (9,2,10) are clustered together. Similarly, 7 is a perturbation of 1, 6 is a perturbation of 3, and 8 is a perturbation of 5. The only original item that was not perturbated is 4, and that is the one item for which the clustering algorithm found no similarity.

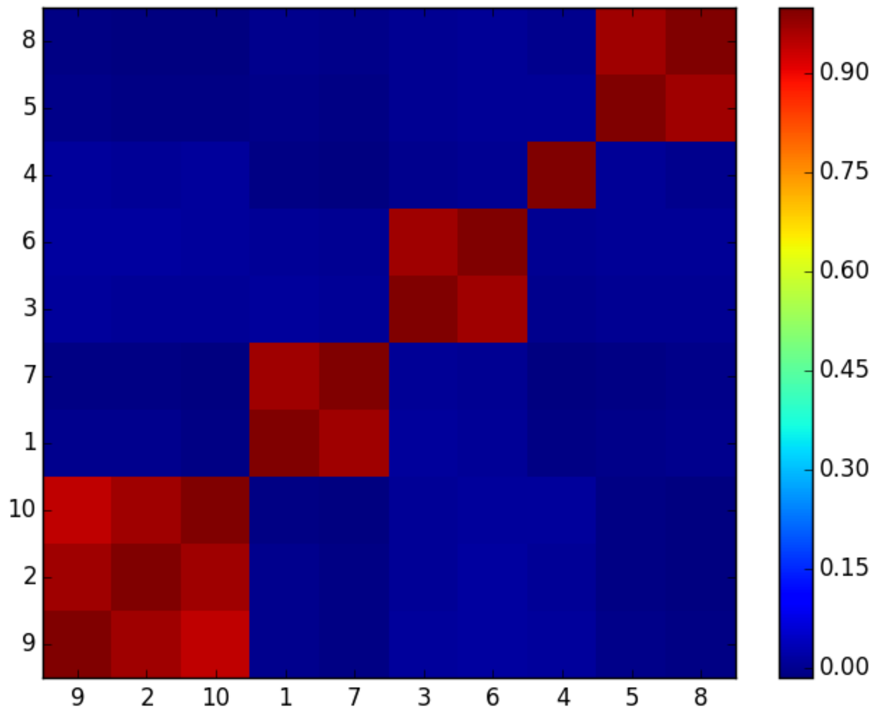

Exhibit 6 – Clustered covariance matrix

Stage 2 quasi-diagonalizes the correlation matrix, in the sense that the largest values lie along the diagonal. However, unlike PCA or similar procedures, HRP does not require a change of basis. HRP solves the allocation problem robustly, while working with the original investments.

Weight #	CLA	HRP	IVP
1	14.44%	7.00%	10.36%
2	19.93%	7.59%	10.28%
3	19.73%	10.84%	10.36%
4	19.87%	19.03%	10.25%
5	18.68%	9.72%	10.31%
6	0.00%	10.19%	9.74%
7	5.86%	6.62%	9.80%
8	1.49%	9.10%	9.65%
9	0.00%	7.12%	9.64%
10	0.00%	12.79%	9.61%

Exhibit 7 – A comparison of three allocations

A characteristic outcome of the three methods studied: CLA concentrates weights on a few investments, hence becoming exposed to idiosyncratic shocks. IVP evenly spreads weights through all investments, ignoring the correlation structure. This makes it vulnerable to systemic shocks. HRP finds a compromise between diversifying across all investments and diversifying across cluster, which makes it more resilient against both types of shocks.

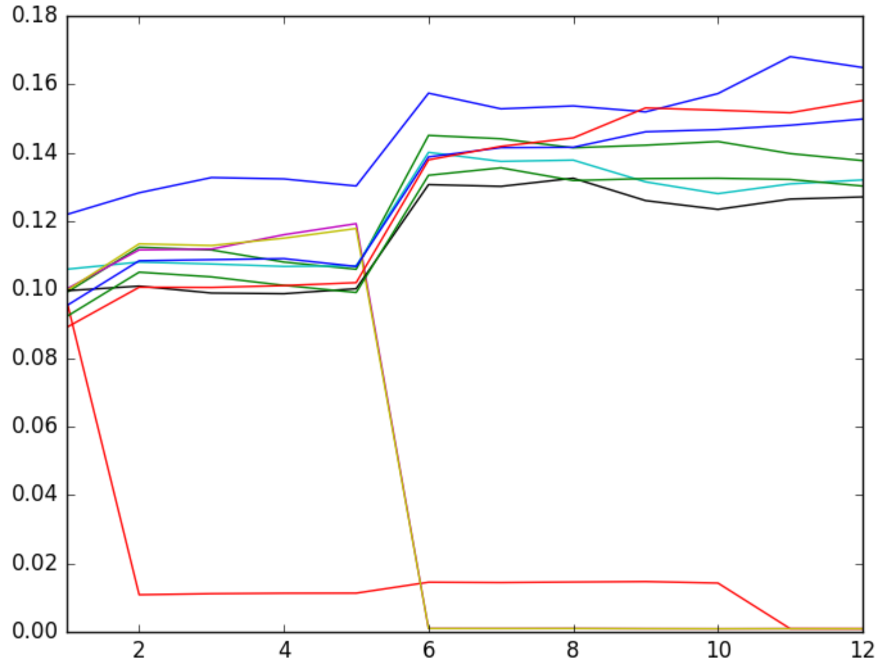

Exhibit 8(a) – Time series of allocations for IVP

Between the first and second rebalance, one investment receives an idiosyncratic shock, which increases its variance. IVP's response is to reduce the allocation to that investment, and spread that former exposure across all other investments. Between the fifth and sixth rebalance, two investments are affected by a common shock. IVP's response is the same. As a result, allocations among the seven unaffected investments grow over time, regardless of their correlation.

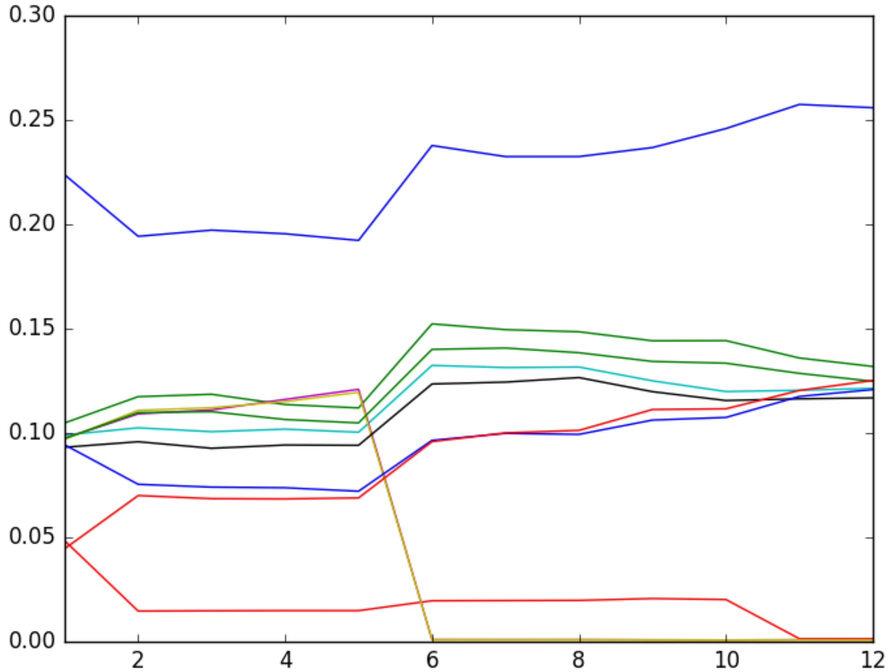

Exhibit 8(b) – Time series of allocations for HRP

HRP's response to the idiosyncratic shock is to reduce the allocation to the affected investment, and use that reduced amount to increase the allocation to a correlated investment that was unaffected. As a response to the common shock, HRP reduces allocation to the affected investments and increases allocation to uncorrelated ones (with lower variance).

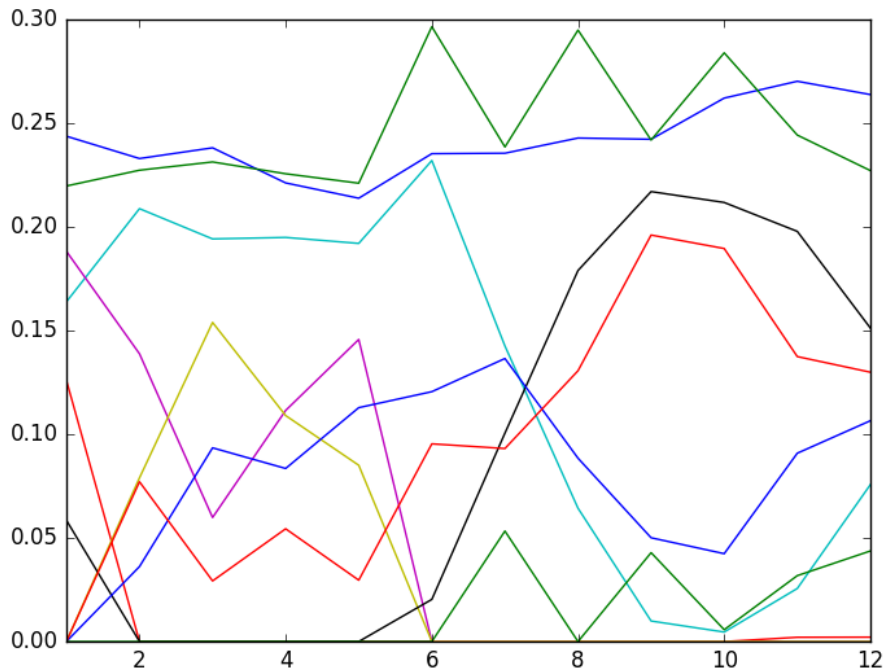

Exhibit 8(c) – Time series of allocations for CLA

CLA allocations respond erratically to idiosyncratic and common shocks. If we had taken into account rebalancing costs, CLA's performance would have been very negative.

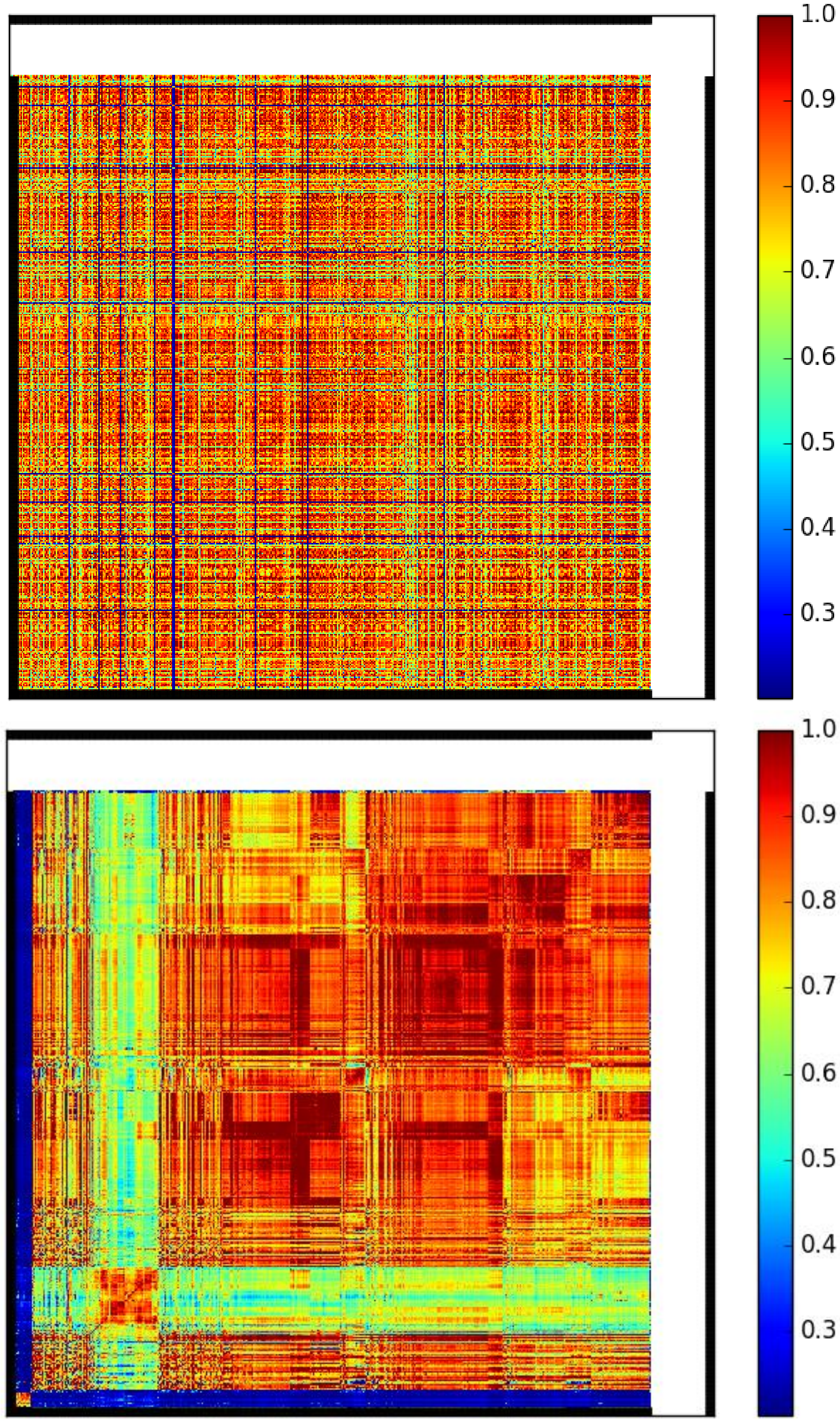

Exhibit 9 – Correlation matrix before and after clustering

The methodology described in this paper can be applied to problems beyond optimization. For example, a PCA analysis of a large fixed income universe suffers the same drawbacks we described for CLA. Small-data techniques developed decades and centuries ago (factor models, regression analysis, econometrics) fail to recognize the hierarchical nature of financial Big Data.

REFERENCES

Bailey, D. and M. López de Prado. “Balanced Baskets: A new approach to Trading and Hedging Risks.” Journal of Investment Strategies, Vol. 1, No. 4 (2012), pp. 21-62. Available at http://ssrn.com/abstract=2066170

Bailey, D. and M. López de Prado. “An open-source implementation of the critical-line algorithm for portfolio optimization.” Algorithms, Vol. 6, No. 1 (2013), pp. 169-196. Available at http://ssrn.com/abstract=2197616

Bailey, D., J. Borwein, M. López de Prado and J. Zhu. “Pseudo-Mathematics and Financial Charlatanism: The Effects of Backtest Overfitting on Out-Of-Sample Performance.” Notices of the American Mathematical Society, Vol. 61, No. 5 (2014), pp. 458-471. Available at http://ssrn.com/abstract=2308659

Bailey, D. and M. López de Prado. “The Deflated Sharpe Ratio: Correcting for Selection Bias, Backtest Overfitting and Non-Normality.” Journal of Portfolio Management, Vol. 40, No. 5 (2014), pp. 94-107.

Black, F. and R. Litterman. “Global portfolio optimization.” Financial Analysts Journal, Vol. 48 (1992), pp. 28–43.

Brualdi, R. The Mutually Beneficial Relationship of Graphs and Matrices. Conference Board of the Mathematical Sciences, Regional Conference Series in Mathematics, Nr. 115 (2010).

Calkin, N. and M. López de Prado. “Stochastic Flow Diagrams.” Algorithmic Finance, Vol. 3, No. 1 (2014), pp. 21-42. Availble at http://ssrn.com/abstract=2379314

Calkin, N. and M. López de Prado. “The Topology of Macro Financial Flows: An Application of Stochastic Flow Diagrams.” Algorithmic Finance, Vol. 3, No. 1 (2014), pp. 43-85. Available at http://ssrn.com/abstract=2379319

Clarke, R., H. De Silva, and S. Thorley. “Portfolio constraints and the fundamental law of active management.” Financial Analysts Journal, Vol. 58 (2002), pp. 48–66.

De Miguel, V., L. Garlappi and R. Uppal, R. “Optimal versus naive diversification: How inefficient is the 1/N portfolio strategy?” Review of Financial Studies, Vol. 22 (2009), pp. 1915–1953.

Jurczenko, E. Risk-Based and Factor Investing. Elsevier Science (2015).

Kolm, P., R. Tutuncu and F. Fabozzi. “60 years of portfolio optimization.” European Journal of Operational Research, Vol. 234, No. 2 (2010), pp. 356-371.

Kuhn, H. W. and A. W. Tucker. "Nonlinear programming". Proceedings of 2nd Berkeley Symposium. Berkeley: University of California Press (1952). pp. 481–492.

Markowitz, H. “Portfolio selection.” Journal of Finance, Vol. 7 (1952), pp. 77–91.

Merton, R. "Option pricing when underlying stock returns are discontinuous". Journal of Financial Economics, Vol. 3 (1976). pp. 125–144.

Michaud, R. Efficient asset allocation: A practical guide to stock portfolio optimization and asset allocation. Boston: Harvard Business School Press (1998).

Ledoit, O. and M. Wolf. “Improved Estimation of the Covariance Matrix of Stock Returns With an Application to Portfolio Selection.” Journal of Empirical Finance, Vol. 10, No. 5 (2003), pp. 603-621.

Rokach, L. and O. Maimon. "Clustering methods", in Data mining and knowledge discovery handbook. Springer, U.S. (2005). pp. 321-352.