Trading with the `portfolio` package

by Jeff Enos, Daniel Gerlanc, and David Kane

October 31, 2008

Abstract

Given a set of current holdings and a target portfolio, that is, a set of desirable holdings to which we would be willing to switch if trading were free, and that our reasons for trading can be captured with one or more rank orderings, the `portfolio` package provides a way to use multiple measures of desirability to determine which trades or portions of trades to do.

1 Introduction

What should we trade now? This question is much more difficult than it might first appear, and yet thousands of individuals and firms controlling trillions of dollars must answer it each day. Consider a simple example.

Imagine that the investment universe is restricted to 10 securities and that our portfolio must hold 5 equal-weighted long positions. At any given point in time, we will hold one of those portfolios. The simplest possible "trade" is to do nothing, keeping the same portfolio in the next period that we hold in the current one. A period can be 5 minutes or 5 months or any length of time. The next simplest trade is a single position swap. Trade one of our 5 current holdings for one of the 5 securities not in the portfolio. There are 25 such trades. Continuing up the complexity scale, there are 100 trades in which we replace 2 securities in the portfolio with 2 securities not in the portfolio. Considering all sets of possible trades, there are 252 options (including the option of no trading), which is equal to the total number of possible portfolios, $\binom{10}{5}$.

In a world of perfect information, we would know the future returns for each of the 10 securities in the universe. Given this information, and some preferences with regard to risk and return, we could examine all 252 options and determine which was best. Unfortunately, in a real world example with thousands of securities in the universe and possibly hundreds in the portfolio, there is no way to forecast results for each possible combination.

These difficulties are exacerbated by other complications.

- **Ranking Trades**: We may have multiple criteria for ranking the trades. Some criteria may be more appropriate for ranking certain types of trades under specific circumstances. In the case where we have a large number of criteria, how do we choose the most appropriate criterion for each trade?

- **Liquidity**: Even if it were simple to determine the ideal portfolio, it may be difficult to get there. Imagine that moving to the ideal portfolio requires that we trade one million shares of IBM; however, IBM typically trades 100,000 shares a day. How are we going to buy all the necessary shares in one day? Even if we bought the entire day's volume (an impossibility) it would take we ten days to get the entire position.

- **Trade Costs**: Trading is not free so we will want to do less of it in the real world than we might care to do in the theoretical world. Basic trading costs (including commissions and spread) tend to enter the calculation linearly. Trade twice as much and we pay twice the costs.

- **Turnover**: Turnover is the flip-side of holding period. In an ideal world, holding period would be endogenous. We would select the holding period which maximised the risk-adjusted return of the portfolio. But, in the real world, almost all portfolios have targeted holding periods that should be adhered to. We are only allowed a certain amount of turnover.

- **Price Impact**: Although commission/spread may be linear in trade volume, price impact is not. We are a participant in the market, and every time we trade we affect the price. Price impact is generally small if we trade a modest portion, say 10%, of volume. But if we trade more, then the price will move against we. Over some range, price impact increases more than linearly.

None of these problems is impossible to overcome, but all of them conspire to make a general solution to the trading problem extremely difficult. Therefore, we simplify.

The `portfolio` package makes three major simplifying assumptions. The first is that we have created a "target" or "ideal" portfolio, a set of positions that is desirable and to which we would be willing to switch if trading were free. This assumption is implausible but it does serve to make the problem tractable. If we only consider trades which move us closer to the target portfolio, it is much easier to handle the other difficulties associated with turnover, liquidity and the like. Instead of looking at all possible buys, for example, we only need to analyse buys for securities in which the target portfolio has more shares than current holdings. The second simplifying assumption is that different criteria for trading can be captured with a rank ordering. We discard the information used to create the ranks. The third simplifying assumption is that no one type of trade is intrinsically better than another type of trade. All things equal, buys, sells, covers, and shorts are equally good.

We divide the document into three sections. First, we create a trade list. Second, we create a trade list in R. Third, we create a long-short trade list in R.

2 A simple example

Assume that we already have a small portfolio consisting of positions in various equities. We have been given an additional $1,000 to invest in the portfolio, and we must invest this $1,000 over the course of one trading day. This is not a realistic scenario, but having a set amount of time in which to trade will simplify our example. Throughout the document, we will refer to our present holdings as the "current" portfolio. The "target portfolio" is an ideal set of holdings that we would immediately switch to if trading were free as per the first simplifying assumption. Note that in this simple example the only trades we will be considering are buys. In subsequent we will introduce more complexity in the form of buys, sells, covers, and shorts.

2.1 Current and target holdings

Our current portfolio consists of shares of 3 companies, IBM (International Business Machines), GM (General Motors) and EBAY (EBay).

```
     shares price
IBM     100    10
GM      100    30
EBAY     75   120
```

The `shares` column expresses how many shares of each stock are in the portfolio, and the `price` column expresses the most recent price of that equity.[1] The market value of the current portfolio can be calculated by summing the products of the shares and prices.

As per the simplifying assumption, we provide a target portfolio.

```
     shares price
GOOG     50    20
EBAY     75   120
IBM     100    10
GM      200    30
SCHW    100    50
MSFT    100    60
```

We would like to buy more shares of GM and take positions in SCHW (Charles Schwab Inc.), MSFT (Microsoft), and GOOG (Google). The market value of the target portfolio is $28,000.

2.2 Portfolio difference

The portfolio difference may be understood as the trades that would change our current holdings into our ideal holdings. If trading were free, we would immediately complete all these trades and reach our target portfolio. Alas,

[1] For simplicity, we use US dollars.

trading is not free, and we will most likely not complete all the orders in one day. Some of them probably require that we purchase a large portion of the daily trading volume (over 10%), at which point the trade may become significantly less desirable.

From the portfolio difference, we determine our *candidate trades*.

candidate trades: The complete set of trades we would have to make to trade from our current portfolio to the target portfolio. If trading were free, we would make all of these trades right now.

Below, we list the candidate trades.

	side	shares	mv
GM	B	100	3000
GOOG	B	50	1000
MSFT	B	100	6000
SCHW	B	100	5000

The `side` column expresses what type of trade we will be making.[2] All the candidate trades are buys so the `side` column only contains `B`. The `shares` column expresses the number of shares of each stock we must buy to reach the target portfolio. The `mv` column expresses the effect that candidate will have on the value of the portfolio. Buys, which increase the value of our portfolio, have a positive value. Sells, which decrease the value of the portfolio, have a negative value.

As the market value of the target portfolio ($28,000) is greater than the market value of the original portfolio ($13,000), we would have to invest an additional $15,000 to trade from our current portfolio to our target portfolio. However, we only have $1,000 with which we may buy additional shares. Therefore, we have to decide which subset of the candidate trades we will make.

2.3 Expressing preferences amongst trades

Part of our simplifying assumption is that we would instantly switch to the target portfolio if trading were free. This implies that all of the candidate trades are desirable. However, they are not all equally desirable. Some trades are better than others. We want to determine which candidate trades or subsets of the candidate trades yield the most utility on the margins.

If we had unlimited funds or could freely trade between our current and target portfolios, we would not have to express preferences amongst trades. However, in the real world, we must decide, given a set of possible trades, which trades we should make first. One way to do this involves assigning each trade a value of overall desirability, *alpha*. When we associate trades with alpha, we say that we sort by alpha or use alpha as a *sort*.

[2] In later examples, S will represent a sell, X will represent a short and C will represent a cover.

alpha: a value, most likely generated by some sort of quantitative model, which expresses the relative quality of the candidate trades. Serves to determine from which candidate trade we would derive the most utility.

sort: a set of values associated with a set of positions. Higher values suggest positive future performance and negative values suggest poor future performance. Based on a sort we can determine what type of trade is most desirable for each position. Therefore, we want to buy or cover positions with positive sort value and sell or short positions with negative sort values.

Like portfolio construction, alpha generation is beyond the scope of this document; we provide alpha values and will be using alpha as a sort. We associate good buys with greater alpha values. In the table below, the candidate at the head of the data frame has the highest alpha value and is therefore the most desirable trade.

```
     side alpha
MSFT    B   1.5
SCHW    B   1.2
GOOG    B   1.1
GM      B   1.0
```

Based on the alpha values, MSFT is the best trade, SCHW is the second best trade, and GM is the worst trade with an alpha value of 1.

2.4 Preliminary ranks

We determine which trades are most desirable by generating an overall measure of desirability for each trade. The first step in generating this value involves creating a *rank ordering* of the trades for each sort we have created. A definition of this term follows:

rank ordering: a linear, relational ordering of the candidates, where each candidate is assigned a rank from the set $1, 2, 3 \ldots n$ where n is the number of candidate trades. Trade 1 provides the greatest utility and trade n provides the least utility. In creating a rank ordering we discard cardinal information such as alpha or one-day return and replace it with a whole number ranking.

We rank and order the candidates by alpha below:

```
     rank side alpha shares   mv
MSFT    1    B   1.5    100 6000
SCHW    2    B   1.2    100 5000
GOOG    3    B   1.1     50 1000
GM      4    B   1.0    100 3000
```

Notice that a rank ordering abstracts the underlying data. Whereas alpha provides an absolute measure of desirability, rank expresses the relative desirability amongst trades. We say that we lose *cardinal information* when we use ranks.

cardinal information: The values used to create a rank ordering. The creation of ranks abstracts these values and replaces them with an ordering that reflects the value of an element relative to other elements in the rank ordering.

In some cases we may want to use more than one measure of desirability. We may have more than one source of cardinal information. Imagine that we want to use both alpha and one-day return as sorts. If we believe in one day reversal, we would assign higher ranks to both orders to sell stocks with positive one-day returns and to orders to buy stocks with negative one-day returns. However, we associate more desirable buys with greater sort values. To account for this, we have taken the inverse of all one-day return values. Therefore, if the return one-day return for GM appears to be 0.10, it should really be −0.10. The table below and to the left shows the one-day return values after we have taken their inverse. The table to the right contains that actual one-day return values.

	rank	side	ret.1.d
GM	1	B	0.10
GOOG	2	B	0.01
MSFT	3	B	-0.01
SCHW	4	B	-0.02

	rank	side	ret.1.d
GM	1	B	-0.10
GOOG	2	B	-0.01
MSFT	3	B	0.01
SCHW	4	B	0.02

GM has the highest rank according to one-day return because it has the most negative return of all the buys, but for the `portfolio` package to properly process our one-day reversal sort, we must take the inverse of all the one-day return values.

2.4.1 The problem of multiple sorting criteria

When we combine the sorts in a single data frame, it is not clear which sort values we should use. If we order by alpha we get the following set of ranks:

	rank	alpha	ret.1.d
MSFT	1	1.5	-0.02
SCHW	2	1.2	-0.01
GOOG	3	1.1	0.01
GM	4	1.0	0.10

Ordering by one-day return yields another ordering:

	rank	alpha	ret.1.d
GM	1	1.0	0.10
GOOG	2	1.1	0.01
SCHW	3	1.2	-0.01
MSFT	4	1.5	-0.02

When we use multiple sorts, there is no obvious way in which we would order the trades by desirability. When sorting by alpha, MSFT is the most desirable trade, and when sorting by one-day return, GM is the most desirable trade. We cannot easily compare or combine the two sorts because we do not know what the exact relationship is between one-day reversal and alpha. Neither sort is even on the same numeric scale. Should we alternate between using values from alpha and one-day return? How would we decide how often to alternate between the sorts? The way in which we express preferences amongst trades in the `portfolio` package represents our answer to these questions. In sections 2.5 through 2.6 we discuss our method for ranking trades when we multiple measures of desirability. In section 2.8 we will discuss the reasoning behind our method of ordering trades.

2.5 Weighting sorts

As sorts express preferences amongst trades, weights express preferences amongst sorts. By assigning each sort a weight, we express how important that sort is relative to other sorts. To illustrate some weighting examples, let's consider the scenario in which we have assigned a weight of 1 to both the alpha and one-day return sorts. By assigning the same weight to both sorts we assert that they are equally important. Assigning a weight directly affects the sort rankings by causing them to be divided by the weight. However, we have assigned both the sorts a weight of 1 so the ranks remain the same.

	rank	side	alpha	shares	mv
MSFT	1	B	1.5	100	6000
SCHW	2	B	1.2	100	5000
GOOG	3	B	1.1	50	1000
GM	4	B	1.0	100	3000

However, the ranks for one-day return remain the same because one-day return has a weight of 1.

	rank	side	ret.1.d	shares	mv
GM	1	B	0.10	100	3000
MSFT	2	B	-0.02	100	6000
SCHW	3	B	-0.01	100	5000
GOOG	4	B	0.01	50	1000

Having divided the original, *raw ranks* by weight, we now have *weighted ranks*.

raw ranks: the original, linearly spaced ranks, built on the scale $1, 2, 3 \ldots n$

weighted ranks: the raw ranks divided by the weights of the sorts.[3]

We now have two ranks associated with each candidate, one from the alpha sort and another from the one-day return sort. To illustrate that we have duplicate ranks for each sort, we bind the equally-weighted alpha and one-day return sorts to form a single data frame.

```
             rank    sort side shares   mv
MSFT.alpha      1   alpha    B    100 6000
GM.ret.1.d      1 ret.1.d    B    100 3000
SCHW.alpha      2   alpha    B    100 5000
MSFT.ret.1.d    2 ret.1.d    B    100 6000
GOOG.alpha      3   alpha    B     50 1000
SCHW.ret.1.d    3 ret.1.d    B    100 5000
GM.alpha        4   alpha    B    100 3000
GOOG.ret.1.d    4 ret.1.d    B     50 1000
```

The row names express the equity ticket symbols, and the string following the period is the name of the sort that generated this rank. At each rank there are two candidates, one of which has been associated with a rank from alpha and the other which has been associated with a rank from one-day return. In cases such as this where we have equally weighted sorts there will be a candidate trade from each sort at every rank.

If we use n sorts, we will have n ranks associated with each candidate. We only want one rank associated with each candidate. So that each candidate only has one rank associated with it, we assign each rank the best rank generated for it by any sort. We have done this in the data frame below.

```
     rank shares   mv
GM      1    100 3000
MSFT    1    100 6000
SCHW    2    100 5000
GOOG    3     50 1000
```

Both GM and MSFT have been assigned a rank of one. This occurs because MSFT has been ranked 1 by the alpha sort and GM has been ranked 1 by the one-day return sort. SCHW has been ranked 2 by the alpha sort and GOOG has been ranked 3 by the alpha sort.

When we equally weight the sorts we are equally likely to use ranks from one either sort. This behaviour is logical because assigning sorts equal weights suggests that they are equally important. However, the sorts may not always be equally important. In the next example we use a weighting scheme that causes us to use one sort to the exclusion of the other.

Let's say that we do not want to consider one-day return. To ignore all of the one-day return values, we make alpha 10 times more important than one-day return. Therefore, we will consider 10 ranks from alpha for every one rank

[3] Note that we are using the term ranks in several different contexts, a possibly abusive notation.

from one-day return. As there are only 4 candidate trades, we will choose the rankings in alpha over the ranks in the one-day return sort.

```
              rank side shares   mv
MSFT.alpha     0.1    B    100 6000
SCHW.alpha     0.2    B    100 5000
GOOG.alpha     0.3    B     50 1000
GM.alpha       0.4    B    100 3000
GM.ret.1.d     1.0    B    100 3000
MSFT.ret.1.d   2.0    B    100 6000
SCHW.ret.1.d   3.0    B    100 5000
GOOG.ret.1.d   4.0    B     50 1000
```

Creating this unbalanced weighting causes us to stack the alpha ranks on top of the one-day return ranks. Since we always assign the lowest rank from all trades to a sort, we will consider the alpha ranks before any of the one-day return ranks.

```
     rank  sort shares   mv
MSFT  0.1 alpha    100 6000
SCHW  0.2 alpha    100 5000
GOOG  0.3 alpha     50 1000
GM    0.4 alpha    100 3000
```

Making the alpha 10 times as important as the one-day return sort causes us to only use ranks from the alpha sort. We do not even consider the number 1 ranked one-day return trade until we examine all the alpha sorts ranked in the top ten. As we only have 4 candidate trades, we do not consider any trades from one-day return.

The last weighting we will consider falls somewhere in between the previous two.

```
             rank side shares   mv
MSFT.alpha   0.67    B    100 6000
GM.ret.1.d   1.00    B    100 3000
SCHW.alpha   1.33    B    100 5000
GOOG.alpha   2.00    B     50 1000
MSFT.ret.1.d 2.00    B    100 6000
GM.alpha     2.67    B    100 3000
SCHW.ret.1.d 3.00    B    100 5000
GOOG.ret.1.d 4.00    B     50 1000
```

To weight the alpha sort by an additional 50%, we divide all the ranks in alpha by 0.5. This causes us to consider 3 ranks from the alpha sort for every 2 rank from the one-day return sort.

```
     rank shares   mv
MSFT 0.67    100 6000
GM   1.00    100 3000
SCHW 1.33    100 5000
GOOG 2.00     50 1000
```

We use three of the ranks from the alpha sort and one rank from the one-day return sort. This is the weighting scheme that we will use in the rest of the example.

We use three of the ranks from the alpha sort and one rank from the one-day return sort. This is the weighting scheme that we use in the rest of the example.

To review, the ranking process has four steps. First, we ranked each trade according to both alpha and one-day return to generate raw ranks. Second, we weight these ranks. Third, we combined the alpha and one-day return ranks. Fourth, we eliminated duplicates by associating each trade with the lowest rank assigned to it by either alpha or one-day return. We call these ranks preliminary ranks because they are not the final values we use to determine the desirability of each trade. Nonetheless, we must generate preliminary ranks before we can arrive at final ranks, the calculation of which we describe in the next section.

2.6 Generating synthetic ranks

If trade 1 is X better than the trade 2, then is trade 99 X better than trade 100? Most portfolio managers would argue that the difference in utility between trade 1 and trade 2 is greater than the difference in utility between trade 99 and trade 100. However, with raw ranks, we make no assertion of how much better one trade is than another trade. To express the tendency for us to derive more utility from the most highly ranked trades, we synthesise yet another set of values from the weighted ranks. We call these values synthetic ranks.[4]

synthetic ranks: values generated by mapping the weighted ranks to a truncated normal distribution ($> 85^{th}$ percentile on $N(0, 1)$).

First, we re-rank the weighted ranks:

```
     rank shares   mv
MSFT    1    100 6000
GM      2    100 3000
SCHW    3    100 5000
GOOG    4     50 1000
```

Next, we evenly distribute the ranks on the interval interval $[0.85, 1)$ such that the best ranked trades are closest to 1 and the worst ranked trades are closest to 0.85:

```
     rank shares   mv rank.s
MSFT    1    100 6000   0.97
GM      2    100 3000   0.94
SCHW    3    100 5000   0.91
GOOG    4     50 1000   0.88
```

We list the scaled ranks in `rank.s`. Next, we map to the a truncated normal distribution.[5]

[4] We are abusing the term "ranks" by using it in several different contexts.

[5] $> 85^{th}$ percentile of $N(0, 1)$

```
     rank rank.s rank.t shares   mv
MSFT    1   0.97    1.9    100 6000
GM      2   0.94    1.6    100 3000
SCHW    3   0.91    1.3    100 5000
GOOG    4   0.88    1.2     50 1000
```

The `rank.t` column lists the ranks mapped to the truncated normal distribution. GM has the best rank and SCHW, has the lowest rank. We might expect to see a `rank.t` of approximately 3.5 for the best ranked trade, but because we only have 4 candidates and the scaled values are evenly spaced on the interval $[0.85, 1)$, the normalised value of the best ranked trade is not as great as it would be if we had 100 trades.

Recall that synthetic ranks express the tendency for there to be greater differences in utility between adjacent, highly ranked trades $(1, 2, 3 \ldots)$ than between adjacent, poorly ranked trades:

rank	Δ	$N(0,1)$	Δ	$> 85^{th}$ of $(0,1)$	Δ
1	1	3.50	1.17	3.50	0.53
2	1	2.32	0.27	2.96	0.21
3	1	2.05	0.17	2.74	0.13
4	1	1.88	0.13	2.61	0.10
5	1	1.75	-	2.51	-
.	.	.	.	.	.
.	.	.	.	.	.
48	1	0.05	0.03	1.46	0.01
49	1	0.02	0.02	1.45	0.01
50	1	0.00	0.02	1.44	0.01
51	1	-0.02	0.02	1.43	0.01
52	1	-0.05	-	1.41	-
.	.	.	.	.	.
.	.	.	.	.	.
96	1	-1.64	0.11	1.06	0.00
97	1	-1.75	0.13	1.06	0.00
98	1	-1.88	0.17	1.06	0.00
99	1	-2.05	0.27	1.06	0.00
100	-	-2.32	-	-	NA

Table 1: Creating synthetic ranks using a linear distribution, a normal distribution, and a truncated normal distribution. Delta columns express the difference in utility between adjacent trades.

Table 1 expresses the differences amongst distributions we might use to rank 100 trades. The `rank` column contains the raw ranks for the 5 best trades, the 5 middle-ranked trades, and the 5 worst trades. In this example the ranks on $[1, 100]$ are spaced at intervals of one. The alpha difference between every trade is the same. If we use raw rank as a measure of alpha, we derive the same utility from every trade. Trade 1 is one better than trade 2, and trade 99 is one better than trade 100.

The normal distribution column ($N(0,1)$) expresses what happens when we normalise the raw ranks. The normal distribution correctly expresses our belief that there is a large difference in alpha between the best ranked trades. However, use of the normal distribution would incorrectly suggest that there are similarly large alpha differences between the worst trades. We get these results when using the normal distribution because the best and worst ranked trades form the tails of the distribution. We do not want large differences in alpha amongst the worst rank trades. The alpha differences decrease until we reach trade 50, then increase again as we move towards the other tail of the distribution. We want alpha to remain the same on the margin past the 50th trade.

To address the problems of a normal and linear distribution, we use a truncated normal distribution, $> 85^{th}\% of N(0,1)$. In the rightmost delta (Δ) column, the alpha differences between the best ranked trades is over 50 times greater than the alpha differences between the worst ranked trades. Every trade ranked worse than 50 has a similar alpha difference. Although the subset $[0.85, 1)$ is slightly arbitrary, (we could have set the lower extreme to be 0.84, 0.85, or another similar value) it serves our purpose of expressing large differences in alpha and where we find the best buys, at one tail, and small differences in alpha amongst the worst buys.

Recall the steps we have taken towards in generating our final measure of rank, synthetic rank. First, we converted the sort values to raw ranks. Second, we converted the raw ranks to weighted ranks. Third, we scaled the weighted ranks to $[0.85, 1)$ to generate scaled weights. Lastly, we mapped the scaled weights to the truncated normal distribution. By only using the 85^{th} percentile and above, we express our belief that the differences in alpha between the best ranked trades is much greater than the differences in alpha between the worst ranked trades.

If the costs associated with trading any stock, all things being equal, were the same, we would not care about the difference in utility between any trades. We would move down the trade list from best to worst until we match the allotted turnover. However, our trading influences prices and may reduce the desirability of a trade.

2.7 Chunks, synthetic rank, and trade-cost adjustment

We want to know at what point the cost of trading an equity exceeds the utility of trading that equity. In the `portfolio` package, we use synthetic rank to represent utility. Determining the cost of purchasing an additional share is impossible if our smallest trading unit is an entire order so we break each order into *chunks*.

chunk: A portion of a candidate trade.

We break candidate trades into chunks by market value. Each chunk has a market value of approximately $2000:

	side	shares	mv	alpha	ret.1.d	rank.t	chunk.shares	chunk.mv
GM.1	B	100	3000	1.0	0.10	1.6	67	2010
GM.2	B	100	3000	1.0	0.10	1.6	33	990
GOOG.1	B	50	1000	1.1	0.01	1.2	50	1000
MSFT.1	B	100	6000	1.5	-0.02	1.9	33	1980
MSFT.2	B	100	6000	1.5	-0.02	1.9	33	1980
MSFT.3	B	100	6000	1.5	-0.02	1.9	33	1980
MSFT.4	B	100	6000	1.5	-0.02	1.9	1	60
SCHW.1	B	100	5000	1.2	-0.01	1.3	40	2000
SCHW.2	B	100	5000	1.2	-0.01	1.3	40	2000
SCHW.3	B	100	5000	1.2	-0.01	1.3	20	1000

The candidate trades are broken into 10 chunks. The number following the period in the row name expresses the chunk number for that particular equity. The `chunks.mv` column expresses the market value of each chunk. The `chunk.shares` column expresses how many shares each chunk consists of.

2.7.1 Trade-cost adjustment of individual chunks

As we trade greater percentages of the average daily volume, the price of the equity will increase. To reflect this phenomenon, we penalise the synthetic ranks of the chunk as we trade greater percentages of the daily volume. We call this penalty *trade-cost adjustment*.

trade-cost adjustment: Lowering a chunk's rank because of trading volume.

To fix this idea, let's first examine the daily volumes of our candidate trades.[6]

	rank.t	volume	shares
GM	1.6	100	100
GOOG	1.2	2200	50
MSFT	1.9	2600	100
SCHW	1.3	2500	100

The trades we want to make for both MSFT, SCHW, and GOOG involve less than 3% of the daily trading volume. However, we must trade 100% of the daily trading volume of GM. We would probably not be able to purchase all these shares in one day, and even if we could, we would affect prices significantly. Moving into the position over several days would be better.

We use a trade-cost adjustment function to express how increasing trade costs reduce the desirability of candidate trades. To better approximate utility, we penalise synthetic ranks at the chunks level. Doing this allows us to better determine at which point the cost of trading an additional chunk is greater than the utility derived by trading an additional chunk. We perform trade-cost adjustment on the chunks by keeping track of what percentage of the daily

[6] The `volume` column represents some measure of past trading volume such as the average trading volume over the last 30 days. A daily measure of `volume` is not required; we would use whatever measure is natural for the frequency with which we trade.

volume we have traded with each additional chunk. The first chunk to cross the threshold of 15% of the daily trading volume is penalised by a fixed amount. All subsequent chunks are penalised by that amount, and any further chunks that pass 30% or 45% percent of the daily trading volume receive further penalties. The chunks of GM, an illiquid equity, have been trade-cost adjusted.

	side	mv	alpha	ret.1.d	rank.t	chunk.shares	chunk.mv	tca.rank
GM.1	B	3000	1.0	0.10	1.6	67	2010	1.6
GM.2	B	3000	1.0	0.10	1.6	33	990	-4.4
GOOG.1	B	1000	1.1	0.01	1.2	50	1000	1.2
MSFT.1	B	6000	1.5	-0.02	1.9	33	1980	1.9
MSFT.2	B	6000	1.5	-0.02	1.9	33	1980	1.9
MSFT.3	B	6000	1.5	-0.02	1.9	33	1980	1.9

The `tca.rank` column expresses the synthetic rank adjusted for trade-cost. As the only candidate for which we want to purchase more than 15% of the daily trading volume is GM, it is the only candidate for which we trade-cost adjust the chunks. Every chunk of GM beyond the first has been trade-cost adjusted. This will cause us to consider the chunks of other candidate trades before we trade additional chunks of GM:

	side	mv	alpha	ret.1.d	rank.t	chunk.shares	chunk.mv	tca.rank
MSFT.1	B	6000	1.5	-0.02	1.9	33	1980	1.9
MSFT.2	B	6000	1.5	-0.02	1.9	33	1980	1.9
MSFT.3	B	6000	1.5	-0.02	1.9	33	1980	1.9
MSFT.4	B	6000	1.5	-0.02	1.9	1	60	1.9
GM.1	B	3000	1.0	0.10	1.6	67	2010	1.6
SCHW.1	B	5000	1.2	-0.01	1.3	40	2000	1.3
SCHW.2	B	5000	1.2	-0.01	1.3	40	2000	1.3
SCHW.3	B	5000	1.2	-0.01	1.3	20	1000	1.3
GOOG.1	B	1000	1.1	0.01	1.2	50	1000	1.2
GM.2	B	3000	1.0	0.10	1.6	33	990	-4.4

As MSFT is the best ranked candidate and does not receive a trade-cost penalty, we would trade all the shares of MSFT before considering the other candidates.[7] Having completed all the trades of MSFT, we would consider the first chunk of GM, the only chunk which has not been trade-cost adjusted. Subsequently, we would trade all the chunks of SCHW and GOOG, the candidate trades ranked 3 and 4. Lastly, we trade the penalised chunks of GM.

2.7.2 Synthetic rank and trade-cost adjustment of small portfolios

In this example, trade-cost adjustment decreases the desirability of the chunks of GM in a non-trivial way. Although GM is ranked 2nd as a candidate trade, every other candidate trade would be made before we completed all the chunks of GM. When we consider such a small number of trades, we assume that all of

[7] Assuming that derived turnover is greater than the market value of all the candidate trades.

the trades are of approximately equal quality; the difference in utility between candidate trades is fairly small. This occurs because the scaled ranks are evenly distributed on $[0.85, 1)$:

	rank	shares	mv	rank.s
MSFT	1	100	6000	0.97
GM	2	100	3000	0.94
SCHW	3	100	5000	0.91
GOOG	4	50	1000	0.88

When we only have 4 candidates, none of the scaled ranks will be very close to 1, and consequently, none of the synthetic ranks will fall at the extreme tail of the normal distribution:

	rank	shares	mv	rank.s	rank.t
MSFT	1	100	6000	0.97	1.9
GM	2	100	3000	0.94	1.6
SCHW	3	100	5000	0.91	1.3
GOOG	4	50	1000	0.88	1.2

Consequently, the difference in utility between candidate trades will be small when there are few candidate trades. Heuristically, this seems correct because if we are making very few trades, we would most likely derive similar utility from any of them.[8] Therefore, it makes sense for us to trade the other three candidates if the costs associated with trading GM are nontrivial.

2.7.3 Synthetic rank and trade-cost adjustment of large portfolios

Moving away from our example for a moment, imagine that we have a large current and target portfolio, the trade list for which contains 100 candidate trades. When we have a large portfolio, we tend to view the differences in utility between candidates in the manner we described in section **??**. When evenly distribute the scaled ranks on the interval $[0.85, 1)$, we have more ranks at the extreme tail:

	rank	rank.s	rank.t
IBM	1	1.00	3.0
GOOG	2	1.00	2.8
GM	3	1.00	2.6
MS	4	0.99	2.5
SCHW	5	0.99	2.4
MSFT	48	0.92	1.4
T	49	0.92	1.4
CVX	50	0.92	1.4
AET	96	0.86	1.1
AMD	97	0.86	1.1
DELL	98	0.85	1.1
EBAY	99	0.85	1.0
HPQ	100	0.85	1.0

[8]This does not exclude our expressing a preference amongst the sorts.

The row names express the equity ticker symbols. `rank` is the raw rank. `rank.s` is the scaled rank, and `rank.t` is the synthetic rank. The best ranked trade IBM, has a scaled rank value very close to one, , and a synthetic rank close to three. This indicates that the best rank falls at the tail of the normal distribution. The worst ranked candidates not only have low synthetic ranks, but they also have very small differences in synthetic rank. If we trade-cost adjust one of the poorly ranked candidates we will most likely not trade it until we have traded all other candidates not penalised by trade cost adjustment. On the other hand, we would still trade IBM, GOOG, or GM, even if some of the chunks had been trade-cost adjusted:

Here we have a subset of the hypothetical chunk table for the 100 candidate example. For this example, the GOOG candidate has been broken up into 2 chunks and the IBM candidate has been broken up into 4 chunks. The ranks of 2^{nd}, 3^{rd}, and 4^{th} chunks of IBM have been penalised for trade costs. Therefore, we trade the first chunk of IBM, followed by all the chunks of GOOG. Subsequently, we trade the remaining chunks of IBM because the trade-cost adjusted rank of its chunks is still greater than the un-penalised synthetic rank of the next most desirable candidate, GM.

Let's quickly review how we generate the final, synthetic ranks. The preliminary values from which we draw the raw ranks are the sorts we define. In this example, we defined sorts for alpha and one-day return. In creating raw ranks abstract away the underlying values provided by the sorts. At this point, we still have a different set of raw ranks for each sorts. To express preferences amongst the sorts, we apply weights to the sorts. This step yields weighted ranks. From the sets of weighted ranks, we associate with each candidate the best weighted rank from any sort. Next, we scale the buys to the interval $[0.85, 1)$. This step yields scaled ranks. From scaled ranks, we generate synthetic ranks by mapping the scaled ranks to a truncated normal distribution. Next, we break the candidates into chunks and perform trade-cost adjustment as necessary. This yields trade-cost adjusted ranks which are the final measure of a chunks desirability.

2.8 Sorting theory

Chooing the best candidate when we have multiple measures of desirability is difficult. Consider the situation where we must choose ten stocks to trade.

Assuming that we use some type of formula to generate alpha, we might be able to incorporate our other sorts into the formula for alpha. Instead of having alpha and one-day return as distinct sorts, we would only have one sort, alpha, which would also take one-day return into account. For this to work, however, we would have to write a function that accounted for the the ordering of every trade by every sort. Furthermore, this function would have to take into account our preference for certain certain sorts over other sorts. To elaborate on the difficulty of this creating such a function, let us consider the situation where we must choose our ten favourite trades, in no particular order, using the data in the table below.

Table 2 has a row for each of 15 candidates, their alpha and one-day return

symbol	raw rank	alpha	symbol	raw rank	ret.1.d
IBM	1	1.57	HPQ	1	-0.063
MS	2	1.26	SUNW	2	-0.056
EBAY	3	1.24	AET	3	-0.041
CBBO	4	1.21	YHOO	4	-0.036
SCHW	5	1.15	T	5	-0.014
PAYX	6	1.12	CVX	6	-0.011
HAL	7	1.12	GOOG	7	-0.011
AMD	8	1.10	PAYX	8	-0.002
MSFT	9	0.99	CBBO	9	0.003
CVX	10	0.96	HAL	10	0.009
AET	11	0.92	QCOM	11	0.011
HPQ	12	0.81	EBAY	12	0.014
QCOM	13	0.77	SCHW	13	0.029
GOOG	14	0.65	AAPL	14	0.036
YHOO	15	0.64	MS	15	0.041

Table 2: The alpha and one-day returns of candidates suggest different rank orderings. All of the candidates are buys.

values, and the raw ranks we would generate from these values. All of the candidates are buys so greater alpha values are better and lesser one-day return values are better.

One portfolio manager might decide that she wants to make trades based only on alpha. She chooses the top ten trades according to alpha. A second portfolio manager may want to make trades based only on one-day return. She chooses the top ten trades according to one-day return. The third portfolio manager considers both alpha and one-day return and choose her favorite trades by examining both.

Portfolio manager three believes in buying equities which have had price decreases of greater than 4% during the previous trading day. Consequently, she would buy HPQ, SUNW, and AET. She would fill her remaining orders using the top 7 trades according to alpha.

How would the third portfolio manager write a function that expresses her trading preferences? What if some days she acted like the first portfolio manager and on other days like the second portfolio manager? How would she account for a change in preference for one of the sorts?

Our solution allows any of these portfolio managers to express her trading preferences without having to write a function that relates the different measures of desirability. Instead, she would use the weighting function that the `portfolio` package provides. She would examine the trade list created using different weighting schemes and adjust the weights until the utility derived from the last candidate traded was greater than the cost of the first trade *not* made.

For example, the portfolio manager may decide that YHOO is a better reversal trade than the last alpha trade and revise the weighting scheme so that she makes one less alpha trade and one more reversal trade.

symbol	raw rank	alpha	symbol	raw rank	ret.1.d
IBM	1	1.57	HPQ	1	-0.063
MS	2	1.26	SUNW	2	-0.056
EBAY	3	1.24	AET	3	-0.041
CBBO	4	1.21	YHOO	4	-0.036
SCHW	5	1.15	T	5	-0.014
PAYX	6	1.12	CVX	6	-0.011
HAL	7	1.12	GOOG	7	-0.011
AMD	8	1.10	PAYX	8	-0.002
MSFT	9	0.99	CBBO	9	0.003
CVX	10	0.96	HAL	10	0.009
AET	11	0.92	QCOM	11	0.011
HPQ	12	0.81	EBAY	12	0.014
QCOM	13	0.77	SCHW	13	0.029
GOOG	14	0.65	AAPL	14	0.036
YHOO	15	0.64	MS	15	0.041

Table 3: Portfolio manager 3 revises her trading preferences.

What ultimately matters is the last candidate we decide to trade and the first candidate we decide not to trade. By using rank orders instead of underlying values, we do not have to combine the different sorts. Instead, we can express our preferences for different, possibly unrelated criteria through the use of a weighting scheme we provide in `portfolio`.

2.9 Pairing trades

Let us return to discussing trade list construction. In practise, most equity portfolios must be maintained at a specific market value. One logical way to achieve this result would be to pair desirable buys and sells of equal market value, and this is what we do in the `portfolio` package. We call these pairings of buys and sells a swap:

swap: A pairing of a buy and sell or short and cover of similar market market value and desirability.

We have already created the framework to create this swaps; we break the candidates into chunks of similar market value and then rank these chunks individually. If our candidate trades included buys and sells, we would simply match the most desirable buys with the most desirable sells. However, our candidate trades are all buys, and we want to increase the market value of our portfolio by $1,000.

2.9.1 Dummy chunks

If we want to increase the market value of the portfolio, we must buy more than we sell. Therefore, we do not want to pair a buy with a sell. We just want buys. The situation where we just want buys or sells is a special case. The `portfolio` package is structured so that we must also trade in pairs. To work within the package framework we introduce the concept of *dummy chunks*:

dummy chunk: A *fake* buy or sell chunk that we pair with a real buy or sell chunk in situations where we want to increase or decrease the market value of the portfolio.

As our example only contains buys, we have paired every buy with a dummy sell.[9]

```
             tca.rank.enter tca.rank.exit rank.gain
MSFT.1,NA.0             1.9         10000     -9998
MSFT.2,NA.0             1.9         10000     -9998
MSFT.3,NA.0             1.9         10000     -9998
MSFT.4,NA.0             1.9         10000     -9998
GM.1,NA.0               1.6         10000     -9998
SCHW.1,NA.0             1.3         10000     -9999
```

In the table above, the row names express the chunk ticker symbols that form the swap. To the left of the comma is an enter chunk, and to the right of the comma is an exit chunk.[10] The exit chunks all have a symbol `NA.0` because they are dummy sells. The `tca.rank.enter` column expresses the trade-cost adjusted rank of the enter chunk, the buy, and the `tca.rank.exit` column expresses the trade-cost adjusted rank of the exit chunk, the dummy sell. The `rank.gain` column expresses the difference in trade-cost adjusted rank between the enter and the exit, the buy and dummy sell.

We have spent considerable time discussing the generation of all types of ranks for buys, but we have not yet discussed ranking sells. For sells, better ranks are more negative. Therefore, a great sell might have a synthetic rank of -3.5. In section **??**, we discuss how we generate the ranks for the sells. For now, just note that better sells have more negative ranks.

Recall that our goal is to make the trades which yield the most utility. In spending our $1,000, we want to trade the best chunks. So that we make the best buys when increasing the market value of the portfolio, we assign the dummy sells an arbitrarily high rank. In the table above, the dummy sells have a trade-cost adjusted rank of -10,000. We match the best the buys and sells by calculating rank gain. As no real sells will yield the same rank gain that the pairing of buy and a dummy sell yields, we create pairs with all the dummy sells

[9] We only show the head of the swaps table.

[10] Enter chunks are either a buy or short. A buy allows us to take a long position and a short allows us to take a short position. Exit chunks are either sells or covers. A sell allows us to exit a long position and a cover allows us to exit a short position.

before even considering other sells. As there are no sells in this example, all the swaps consist of a buy and a dummy sell.

Let's quickly review why we create swaps. We want to maximise utility by making the candidate trades or portions of candidate trades that yield the greatest utility. Generally, we want to maintain the portfolio equity at a constant level. A logical way to do this involves pairing buys and sells of similar market value. To maximise utility, we should pair the most best ranked buys and sells. In special cases, we want to increase or decrease the market value of our portfolio. In order to do this, we must make more of one type of trade. However, this would require that we have swaps that contain only a buy or sell. Since we cannot have a swap of only one trade, we introduce dummy trades. As dummy trades have an arbitrarily high synthetic rank they pair with the best buys and sells to ensure that we choose the most useful candidates in changing the market value of the portfolio.

2.10 Accounting for turnover

Note: this and subsequent sections need to account for change in turnover application. Now all swaps are done such that the total market value of trades goes up to but doesn't exceed the turnover amount. In the meantime I have adjusted the example's turnover to $2,000 so that at least one chunk is done, although now Sweave chunks will be inconsistent with the text.

As we stated earlier, holding period would be endogenous if we could always set it to maximise risk-adjusted return. However, most real world portfolios have a set holding period and consequently, a set turnover. There is no real concept of turnover or holding period in this example. We have \$1,000 to invest in our portfolio over the course of a single day. Although this additional investment does not represent turnover, we can view our \$1,000 as representing a daily turnover of \$1,000. We want to make the best ranked trades until the cumulative market value of these trades exceeds the money we have to invest. Analogously, we would say that we want to make the best ranked trades until we exceed turnover.

As our turnover in this example is \$2000, all of our trades will not have a market value greater than \$2000:

```
            tca.rank.enter tca.rank.exit rank.gain
MSFT.1,NA.0            1.9         10000     -9998
```

MSFT is the the best ranked trade. Consequently, we choose swaps of MSFT before choosing other swaps. We make 1 because each swap has a value of approximately \$2000, and our turnover is \$2000.

2.11 Actual orders

We do not want to submit two orders for 8 shares of MSFT. Before submitting the trade list, we must roll-up the swaps into larger orders. We first remove the dummy chunks:

```
       side   mv alpha ret.1.d rank.t chunk.shares chunk.mv tca.rank
MSFT.1    B 6000   1.5   -0.02    1.9           33     1980      1.9
```

Then we combine the chunks to form a single order per candidate:

```
     side shares   mv alpha ret.1.d rank.t
MSFT    B     33 1980   1.5   -0.02    1.9
```

We now have an order for 33 shares of MSFT, which is the sum of the chunks of MSFT. Having discussed in words the process of trade list creation, we describe, step-by-step, the process of building a `tradelist` object in R.

3 Creating a long-only tradelist in R

To create a `tradelist`, we need four main pieces. The first two pieces necessary to create a `tradelist` are `portfolio` objects. One of these portfolios is our current portfolio.

Our current portfolio is a superset of the previous holdings. The major difference between the two portfolios is that the current portfolio in this example includes positions that we sell. This `portfolio`, named `p.current`, consists of 6 positions and has a market value of $47,750.

```
> p.current.shares

     shares price
IBM     100    10
GM      100    30
EBAY     75   120
DELL     50   110
QCOM     75   190
AMD     150   100
```

The target portfolio is a superset of the previous target portfolio. It contains 6 positions and has a market value of $47,500.

```
> p.target.shares

     shares price
GOOG     50    20
EBAY     75   120
IBM     100    10
GM      200    30
SCHW    100    50
MSFT    100    60
AMD     100   100
QCOM     50   190
```

We calculate the portfolio difference to determine the candidate trades.[11]

```
> sub.candidates

     orig target side shares    mv
AMD   150    100    S     50 -5000
DELL   50      0    S     50 -5500
GM    100    200    B    100  3000
GOOG    0     50    B     50  1000
MSFT    0    100    B    100  6000
QCOM   75     50    S     25 -4750
SCHW    0    100    B    100  5000
```

The candidate buys are the same as before and we have 3 candidate sells. The market value is signed and expresses the net effect a candidate has on the dollar value of a portfolio.

3.1 Assigning weights

We assign weights to the sorts by creating a list.

```
> sorts <- list(alpha = 1, ret.1.d = 1.1)
```

We assign a weight of 1 to alpha and a weight of 1.1 to one-day return.

3.2 Passing additional information to `tradelist`

The fourth item is a data frame. The `portfolio` package requires that this data frame contain columns for `id`, `volume`, `price.usd`, and the sorts:

```
> sub.data

       id volume price.usd alpha ret.1.d
IBM   IBM   2100        10 -0.76  -0.003
GOOG GOOG   2200        20  1.10   0.010
GM     GM    100        30  1.00   0.100
SCHW SCHW   2500        50  1.20  -0.010
MSFT MSFT   2600        60  1.50  -0.020
AMD   AMD   3000       100 -0.94   0.010
DELL DELL   3100       110 -0.15   0.070
EBAY EBAY   3200       120 -0.32   0.001
QCOM QCOM   3900       190 -0.36  -0.005
```

`volume` expresses some measure of average trading volume. `price.usd` is the most recent price of the security in US dollars. We must also include the sorts we define in `sorts`, `alpha` and `ret.1.d`.

[11] The data frame is a subset of the `candidates` data frame. We often take subsets of data frames so that they fit better on the page. If we do so we indicate this by prepending the name of the data frame with `sub`.

3.3 Calling `new`

We use `p.current`, `p.target`, the `sorts`, and `data` as arguments to `new`.

```
> tl <- new("tradelist", orig = p.current, target = p.target,
+     chunk.usd = 2000, sorts = sorts, turnover = 30250, data = data)
```

In this call, the `new` method for `tradelist` accepts 8 parameters:[12] The first argument, `"tradelist"`, specifies the name of the object that we want to create. The argument to the `orig` parameter, `p.current`, is the current portfolio. The argument to the `target` parameter, `p.current`, is the target portfolio. The `sorts` parameter accepts the `sorts` list we created earlier. We create chunks with a granularity of of $2,000. The `data` parameter accepts the data frame we created earlier with columns for `id`, `volume`, `price.usd`, and the sorts.

The `turnover` parameter accepts an integer argument which expresses the maximum market value all orders made in one session. In the previous example we only had $1,000 with which we could buy stocks. In this example, we can both buy and sell equities. We might sell an equity and use the proceeds to buy another equity. However, the turnover restriction applies to sells just as much as buys. If we have a turnover of $1,000, we may make $1,000 worth of buys, $1,000 worth of sells, or something in between. For this example, we have set the turnover equal to the unsigned market value of all the candidate trades. This means that we take the absolute value of all market values, which is $30,250. Having set `turnover` to this value, we complete every candidate trade.

We have demonstrated how to create a simple `tradelist` in R. In the next section we examine the `tradelist` that we have constructed. In doing so, we learn how the `tradelist` generation algorithm works.

4 The `tradelist` algorithm

The `tradelist` code provides an algorithm, divisible into seven smaller steps, that generates a set of trades that will move the current, original portfolio towards an ideal, target portfolio. The seven steps in the algorithm correspond to the following methods of the `tradelist` class: `calcCandidates`, `calcRanks`, `calcChunks`, `calcSwaps`, `calcSwapsActual`, `calcChunksActual`, and `calcActual`.

The user never needs to directly call any of these methods when using the `portfolio` package. A call to the `new` method of the `tradelist` class invokes the `initialize` method of `tradelist`. The `initialize` method then calls the seven methods serially. The first step of the `tradelist` algorithm involves determining which types of orders we must make in order to trade towards the target portfolio.

4.1 The `calcCandidates` method

As stated in our simplifying assumption, we only consider trades that bring us closer to the target portfolio. To determine candidate trades we calculate which

[12] The `new` method of `tradelist` can accept more parameters, but they are optional.

positions have changed. If a position has changed, we determine what type of trade the candidate is (buy or sell) by taking the portfolio difference to generate a list of candidate trades.

```
> tl@candidates

       id orig target side shares    mv
AMD   AMD  150    100    S     50 -5000
DELL DELL   50      0    S     50 -5500
GM     GM  100    200    B    100  3000
GOOG GOOG    0     50    B     50  1000
MSFT MSFT    0    100    B    100  6000
QCOM QCOM   75     50    S     25 -4750
SCHW SCHW    0    100    B    100  5000
```

Given the data stored in the `candidates` data frame and the `data` data frame, the `portfolio` package can generate the trade list.

4.2 The `calcRanks` Method

Ranking the trades is possibly the most complicated task delegated to the `tradelist` class. When the rank-generating algorithm returns, the `ranks` data frame `tradelist` will contain the synthetic rank, `rank.t`, for each trade.

4.2.1 Interpretation of sort values

When we define a sort, we express our preference for purchasing different stocks. Lesser values express a preference for selling or shorting a position and greater values express a preference for buying or covering a position. In the previous example we only saw positive alpha values because all the candidates were buys. If the values were not positive, we might question why the trade was even a candidate. Recall our first simplifying assumption that all of the candidates are desirable and the `portfolio` package only helps us to determine which are the most desirable.

In real life, we want to create a sort using meaningful values that express our trading preferences. One such value is one-day return.

4.2.2 Creating raw ranks for a long-only portfolio

The first step in creating ranks is generating raw ranks. We break the trades into separate data frames by side and rank the trades within each side because one type of trade is no than another type of trade.

```
$B
       id orig target side shares   mv ret.1.d rank
GM     GM  100    200    B    100 3000    0.10    1
GOOG GOOG    0     50    B     50 1000    0.01    2
SCHW SCHW    0    100    B    100 5000   -0.01    3
MSFT MSFT    0    100    B    100 6000   -0.02    4
```

```
$S
       id orig target side shares    mv ret.1.d rank
QCOM QCOM   75     50    S     25 -4750  -0.005    1
AMD   AMD  150    100    S     50 -5000   0.010    2
DELL DELL   50      0    S     50 -5500   0.070    3
```

The `$B` data frame shows the buys ranked with other buys and the `$S` data frame shows the sells ranked with other sells. The most desirable buys are those associated with the greatest values in `ret.1.d`. The most desirable sells are those associated with the least value in `ret.1.d`. Therefore, GM ranked 1 amongst buys, is the most desirable buy, and QCOM, ranked 1 amongst sells, is the most desirable sell.[13]

4.2.3 Interleaving

We now have two tables of ranks and there are still multiple trades at each rank: a buy and sell ranked number one, number two and so on. Combining the two tables of ranks by type leaves us with duplicates:

```
     orig target side shares    mv ret.1.d rank
GM    100    200    B    100  3000   0.100    1
QCOM   75     50    S     25 -4750  -0.005    1
GOOG    0     50    B     50  1000   0.010    2
AMD   150    100    S     50 -5000   0.010    2
SCHW    0    100    B    100  5000  -0.010    3
DELL   50      0    S     50 -5500   0.070    3
MSFT    0    100    B    100  6000  -0.020    4
```

We argue that there is no natural way to choose between the best buy and best sell. To deal with this ambiguity, we always break ties in rank between a buy and sell by assigning the buy the higher rank. In the following table, we create new raw ranks to eliminate the duplicates.

```
     orig target side shares    mv alpha rank
MSFT    0    100    B    100  6000  1.50    1
AMD   150    100    S     50 -5000 -0.94    2
SCHW    0    100    B    100  5000  1.20    3
QCOM   75     50    S     25 -4750 -0.36    4
GOOG    0     50    B     50  1000  1.10    5
DELL   50      0    S     50 -5500 -0.15    6
GM    100    200    B    100  3000  1.00    7
```

Notice that each candidate has a unique rank and that the rows alternate between buy and sell candidates. The best ranked candidate trade is a buy

[13] We have taken the inverse of all the one-day return values so that the `portfolio` package interprets them correctly. If we believe one-day reversal, the best buys have negative one-day returns and the best sells have positive one-day returns. Buy low, sell high. However, the `portfolio` package interprets greater values as indicative of the best buys and lesser values as indicate of the best sells.

because we broke the tie for first between the best ranked buy and sell by assigning the buy the higher rank. This pattern repeats throughout the data frame because we have ties at every rank except the last. We call this process of alternating between the best ranked buys and sells *interleaving*.

interleaving: The process of breaking the trades up by side and ranking them with other trades of the same type, thereby yielding multiple trades at each rank. We always break ties in rank with the following ordering: Buys, Sells, Covers, Shorts (B, S, C, X).

4.2.4 Weighted ranks

Having interleaved the candidates, we divide the new raw ranks by the weight assigned to one-day return, 1.1.

```
       id orig target side shares    mv ret.1.d rank
GM     GM  100    200    B    100  3000   0.100 0.83
QCOM QCOM   75     50    S     25 -4750  -0.005 1.65
GOOG GOOG    0     50    B     50  1000   0.010 2.48
AMD   AMD  150    100    S     50 -5000   0.010 3.31
SCHW SCHW    0    100    B    100  5000  -0.010 4.13
DELL DELL   50      0    S     50 -5500   0.070 4.96
MSFT MSFT    0    100    B    100  6000  -0.020 5.79
```

We assigned alpha a weight of 1 so the ranks remain the same.

```
> tl@rank.sorts[["alpha"]]

       id orig target side shares    mv alpha rank
MSFT MSFT    0    100    B    100  6000  1.50    1
AMD   AMD  150    100    S     50 -5000 -0.94    2
SCHW SCHW    0    100    B    100  5000  1.20    3
QCOM QCOM   75     50    S     25 -4750 -0.36    4
GOOG GOOG    0     50    B     50  1000  1.10    5
DELL DELL   50      0    S     50 -5500 -0.15    6
GM     GM  100    200    B    100  3000  1.00    7
```

We combine the alpha and one-day return ranks into a single data frame.

```
     id orig target side shares    mv rank
1   AMD  150    100    S     50 -5000  2.0
2   AMD  150    100    S     50 -5000  3.6
3  DELL   50      0    S     50 -5500  6.0
4  DELL   50      0    S     50 -5500  5.5
5    GM  100    200    B    100  3000  7.0
6    GM  100    200    B    100  3000  0.9
7  GOOG    0     50    B     50  1000  5.0
8  GOOG    0     50    B     50  1000  2.7
9  MSFT    0    100    B    100  6000  1.0
10 MSFT    0    100    B    100  6000  6.4
```

```
11 QCOM   75     50    S     25 -4750  4.0
12 QCOM   75     50    S     25 -4750  1.8
13 SCHW    0    100    B    100  5000  3.0
14 SCHW    0    100    B    100  5000  4.5
```

To remove duplicates, we assign each candidate the best weighted rank associated with it by any sort.

```
     orig target side shares    mv
GM    100    200    B    100  3000
MSFT    0    100    B    100  6000
QCOM   75     50    S     25 -4750
AMD   150    100    S     50 -5000
GOOG    0     50    B     50  1000
SCHW    0    100    B    100  5000
DELL   50      0    S     50 -5500
```

And we re-rank the candidates.

```
     target side shares    mv rank.t
GM      200    B    100  3000    1.9
MSFT    100    B    100  6000    1.6
QCOM     50    S     25 -4750   -1.8
AMD     100    S     50 -5000   -1.4
GOOG     50    B     50  1000    1.3
SCHW    100    B    100  5000    1.2
DELL      0    S     50 -5500   -1.2
```

4.2.5 Mapping to the truncated normal distribution

Having weighted the ranks we create synthetic ranks from a truncated normal distribution. When we only have buys, we scale the weighted ranks to $[0.85, 1)$. This gives us the positive tail of the normal distribution. We associate more negative values with better sells so we want to map sells to the negative tail of the normal distribution. To do this, we scale sells to the interval $(0, 0.15]$.

```
     side alpha ret.1.d rank rank.ws
QCOM    S -0.36  -0.005  1.8   0.037
AMD     S -0.94   0.010  2.0   0.075
DELL    S -0.15   0.070  5.5   0.112
SCHW    B  1.20  -0.010  3.0   0.880
GOOG    B  1.10   0.010  2.7   0.910
MSFT    B  1.50  -0.020  1.0   0.940
GM      B  1.00   0.100  0.9   0.970
```

We map the scaled ranks to the normal distribution.

```
> tl.ranks
```

	id	orig	target	side	shares	mv	alpha	ret.1.d	rank.t
QCOM	QCOM	75	50	S	25	-4750	-0.36	-0.005	-1.8
AMD	AMD	150	100	S	50	-5000	-0.94	0.010	-1.4
DELL	DELL	50	0	S	50	-5500	-0.15	0.070	-1.2
SCHW	SCHW	0	100	B	100	5000	1.20	-0.010	1.2
GOOG	GOOG	0	50	B	50	1000	1.10	0.010	1.3
MSFT	MSFT	0	100	B	100	6000	1.50	-0.020	1.6
GM	GM	100	200	B	100	3000	1.00	0.100	1.9

`rank.t` expresses the synthetic rank. All of the sells have a negative `rank.t` because they have been mapped to the negative tail of the normal distribution, while all of the buys have a positive `rank.t` because they have been mapped to the other tail. As described in section 2.7.3, the synthetic ranks do not fall at the extreme tail of the normal distribution.

4.3 The `calcChunks` Method

Having calculated synthetic ranks, the `portfolio` package creates the chunks table. We defined the market value of each chunk by specifying the `chunk.usd` parameter in the call to `new`. The addition of sells does not have a dramatic effect on the manner in which we generate the chunk table besides contributing negative trade-cost adjusted ranks.

```
> sub.chunks
```

	side	rank.t	chunk.shares	chunk.mv	tca.rank
AMD.1	S	-1.4	20	-2000	-1.4
AMD.2	S	-1.4	20	-2000	-1.4
AMD.3	S	-1.4	10	-1000	-1.4
DELL.1	S	-1.2	18	-1980	-1.2
DELL.2	S	-1.2	18	-1980	-1.2
DELL.3	S	-1.2	14	-1540	-1.2
GM.1	B	1.9	67	2010	1.9
GM.2	B	1.9	33	990	-4.1
GOOG.1	B	1.3	50	1000	1.3
MSFT.1	B	1.6	33	1980	1.6
MSFT.2	B	1.6	33	1980	1.6
MSFT.3	B	1.6	33	1980	1.6
MSFT.4	B	1.6	1	60	1.6
QCOM.1	S	-1.8	11	-2090	-1.8
QCOM.2	S	-1.8	11	-2090	-1.8
QCOM.3	S	-1.8	3	-570	-1.8
SCHW.1	B	1.2	40	2000	1.2
SCHW.2	B	1.2	40	2000	1.2
SCHW.3	B	1.2	20	1000	1.2

Most chunks have an unsigned market value of approximately \$2,000. The only chunks of market value significantly less than \$2,000 are the final chunks of a candidate. These chunks are the remainders left after dividing the rest of the order into \$2,000 chunks.

If we order the chunks by `tca.rank`, the second chunk of GM has been severely penalised for trade costs.

```
> head(sub.chunks[order(sub.chunks[["tca.rank"]]), ])

       side rank.t chunk.shares chunk.mv tca.rank
GM.2      B    1.9           33      990     -4.1
QCOM.1    S   -1.8           11    -2090     -1.8
QCOM.2    S   -1.8           11    -2090     -1.8
QCOM.3    S   -1.8            3     -570     -1.8
AMD.1     S   -1.4           20    -2000     -1.4
AMD.2     S   -1.4           20    -2000     -1.4
```

GM has a more negative `tca.rank` than any of the buys or sells, indicating that this is the last chunk we would trade.

4.4 The `calcSwaps` Method

The `calcSwaps` works in as it did in the previous example, the main difference being that we pair real buy chunks with real sell chunks. We determine which trades to pair for a swap by calculating *rank gain*.

rank gain: The difference in `tca.rank` between a buy and a sell. As the most desirable buys have a very positive `tca.rank` and the most desirable sells have a very negative `tca.rank`, the best swaps have great `rank.gain` values.

Buys with high `tca.rank` have been matched with sells with low `tca.rank`.

```
> swaps.sub

              side.enter tca.rank.enter side.exit tca.rank.exit rank.gain
GM.1,QCOM.1            B            1.9         S          -1.8       3.7
MSFT.1,QCOM.2          B            1.6         S          -1.8       3.3
MSFT.2,QCOM.3          B            1.6         S          -1.8       3.3
MSFT.3,AMD.1           B            1.6         S          -1.4       3.0
MSFT.4,AMD.2           B            1.6         S          -1.4       3.0
GOOG.1,AMD.3           B            1.3         S          -1.4       2.8
SCHW.1,DELL.1          B            1.2         S          -1.2       2.4
SCHW.2,DELL.2          B            1.2         S          -1.2       2.4
SCHW.3,DELL.3          B            1.2         S          -1.2       2.4
GM.2,NA.0              B           -4.1         S       10000.0  -10004.1
```

We have paired almost all of the buy chunks with real sell chunks. The only buy we have not paired with a real sell chunk is the second chunk of GM. As the target portfolio ($47,500) has approximately the same market value as the current portfolio ($47,750), we will not introduce any dummy chunks to account for over or under-investment. We pair GM with a dummy chunk only because we have run out of real sell chunks to match it with. As we would rather make swaps which contain a real buy and sell chunk, we assign the dummy sell chunk a poor `tca.rank` which yields a low `rank.gain` value. Consequently, we will not consider this trade until we have considered all of the other trades.

4.5 The `calcSwapsActual` Method

The remaining steps of the `tradelist` algorithm clean up the `tradelist` for final use. In the `calcSwapsActual` method we remove the most poorly ranked swaps that exceed turnover. When we created the `tradelist`, we set `turnover` to be $30,250, the unsigned market value of all the candidate trades. A `turnover` of $30,250 will allow us to complete every trade.

```
> sub.swaps.actual

               side.enter tca.rank.enter side.exit tca.rank.exit rank.gain
GM.1,QCOM.1             B            1.9         S          -1.8       3.7
MSFT.1,QCOM.2           B            1.6         S          -1.8       3.3
MSFT.2,QCOM.3           B            1.6         S          -1.8       3.3
MSFT.3,AMD.1            B            1.6         S          -1.4       3.0
MSFT.4,AMD.2            B            1.6         S          -1.4       3.0
GOOG.1,AMD.3            B            1.3         S          -1.4       2.8
SCHW.1,DELL.1           B            1.2         S          -1.2       2.4
SCHW.2,DELL.2           B            1.2         S          -1.2       2.4
SCHW.3,DELL.3           B            1.2         S          -1.2       2.4
```

Right now, turnover does not cause any swaps to be dropped because it is greater than the unsigned market value of all the candidate trades, which is $30,250.

We can cause some swaps to be dropped by setting `turnover` to a value less than $30,250.

```
> tl@turnover <- 30250 - tl@chunk.usd
```

When we set turnover to a value equal to one chunk less (2000 than the difference in market value between the original and target portfolios, the `calcSwapsActual` method excises the swap with the lowest `tca.rank`.

```
> sub.swaps.actual

               side.enter tca.rank.enter side.exit tca.rank.exit rank.gain
GM.1,QCOM.1             B            1.9         S          -1.8       3.7
MSFT.1,QCOM.2           B            1.6         S          -1.8       3.3
MSFT.2,QCOM.3           B            1.6         S          -1.8       3.3
MSFT.3,AMD.1            B            1.6         S          -1.4       3.0
MSFT.4,AMD.2            B            1.6         S          -1.4       3.0
GOOG.1,AMD.3            B            1.3         S          -1.4       2.8
SCHW.1,DELL.1           B            1.2         S          -1.2       2.4
SCHW.2,DELL.2           B            1.2         S          -1.2       2.4
```

We have removed the third chunk of GM from the list.

4.6 The calcChunksActual Method

Our `tradelist` is almost complete, but first we must change the swaps back into chunks. In addition, we do not want to include any orders for dummy chunks, so we will remove those when we turn the swaps back into chunks.

```
> sub.chunks.actual

       side alpha ret.1.d rank.t tca.rank chunk.shares chunk.mv chunk
GM.1      B  1.00   0.100    1.9      1.9           67     2010     1
MSFT.1    B  1.50  -0.020    1.6      1.6           33     1980     1
MSFT.2    B  1.50  -0.020    1.6      1.6           33     1980     2
MSFT.3    B  1.50  -0.020    1.6      1.6           33     1980     3
MSFT.4    B  1.50  -0.020    1.6      1.6            1       60     4
GOOG.1    B  1.10   0.010    1.3      1.3           50     1000     1
SCHW.1    B  1.20  -0.010    1.2      1.2           40     2000     1
SCHW.2    B  1.20  -0.010    1.2      1.2           40     2000     2
SCHW.3    B  1.20  -0.010    1.2      1.2           20     1000     3
QCOM.1    S -0.36  -0.005   -1.8     -1.8           11    -2090     1
QCOM.2    S -0.36  -0.005   -1.8     -1.8           11    -2090     2
QCOM.3    S -0.36  -0.005   -1.8     -1.8            3     -570     3
AMD.1     S -0.94   0.010   -1.4     -1.4           20    -2000     1
AMD.2     S -0.94   0.010   -1.4     -1.4           20    -2000     2
AMD.3     S -0.94   0.010   -1.4     -1.4           10    -1000     3
DELL.1    S -0.15   0.070   -1.2     -1.2           18    -1980     1
DELL.2    S -0.15   0.070   -1.2     -1.2           18    -1980     2
DELL.3    S -0.15   0.070   -1.2     -1.2           14    -1540     3
       dummy.quality
GM.1            <NA>
MSFT.1          <NA>
MSFT.2          <NA>
MSFT.3          <NA>
MSFT.4          <NA>
GOOG.1          <NA>
SCHW.1          <NA>
SCHW.2          <NA>
SCHW.3          <NA>
QCOM.1          <NA>
QCOM.2          <NA>
QCOM.3          <NA>
AMD.1           <NA>
AMD.2           <NA>
AMD.3           <NA>
DELL.1          <NA>
DELL.2          <NA>
DELL.3          <NA>
```

All of the dummy chunks have been removed.

4.7 The Final Step: Actual Orders

In the last step of `tradelist` generation, we "roll-up" the actual chunks for each security to form one order per security.

```
> tl.actual

     side shares    mv alpha ret.1.d rank.t
AMD     S     50 -5000 -0.94   0.010   -1.4
DELL    S     50 -5500 -0.15   0.070   -1.2
GM      B     67  2010  1.00   0.100    1.9
GOOG    B     50  1000  1.10   0.010    1.3
MSFT    B    100  6000  1.50  -0.020    1.6
QCOM    S     25 -4750 -0.36  -0.005   -1.8
SCHW    B    100  5000  1.20  -0.010    1.2
```

No rows for chunks remain in the `actual` data frame.

5 A Long-Short Example

For the most part, the `portfolio` package treats one-sided and long-short portfolios similarly. The major difference is that we now have to take four types of trades into consideration, buys, sells, shorts, and covers.

5.1 Current and target portfolios

Our current portfolio is a superset of the holdings in the previous example. This example's current portfolio includes positions that we will short and cover. The current portfolio, `p.current`, consists of 11 positions and has a market value of $16,780.

```
> p.current.shares

     shares price
IBM     100    10
GM      100    30
AMD     150   100
DELL     50   110
EBAY     75   120
QCOM     75   190
HPQ     -50    15
HAL     -75    20
PAYX   -100    25
TXN     -25    25
YHOO    -10    20
```

The target portfolio is a superset of the target portfolio we used in the two previous examples. It contains all the positions in the previous target portfolio plus positions that we short or cover.

```
> p.target.shares

     shares price
IBM     100    10
GOOG     50    20
GM      200    30
SCHW    100    50
MSFT    100    60
AMD     100   100
EBAY     75   120
QCOM     50   190
HPQ    -100    15
HAL     200    20
PAYX    -50    25
APPL    -75    30
TXN     -50    25
YHOO     25    20
```

The target portfolio, `p.target`, contains 14 positions and has a market value of $44,900. We assume that we have the additional funds necessary to increase the market value of the portfolio.

5.2 Candidate trades

We calculate the portfolio difference to determine what the candidate trades will be:

```
> sub.candidates

     orig target side shares    mv
AMD   150    100    S     50 -5000
APPL    0    -75    X     75 -2250
DELL   50      0    S     50 -5500
GM    100    200    B    100  3000
GOOG    0     50    B     50  1000
HAL   -75      0    C     75  1500
HPQ   -50   -100    X     50  -750
MSFT    0    100    B    100  6000
PAYX -100    -50    C     50  1250
QCOM   75     50    S     25 -4750
SCHW    0    100    B    100  5000
TXN   -25    -50    X     25  -625
YHOO  -10      0    C     10   200
```

We now have buy, sell, cover, and short candidates (B, S, C, X). Buys and covers have positive market values because they increase the value of the portfolio, and sells and shorts have negative market values because they decrease the value of the portfolio. Notice that all the candidate trades necessary to reach the target positions for HAL and YHOO are not on the candidate list. We do not include all the candidate trades to reach these positions because they involve side changes.

5.2.1 Side changes and restrictions

A side change occurs when a position changes from long to short or short to long. The `portfolio` package does not allow a side change to occur during a single trading session.[14] For a side change to occur, we must make two types of trades. We must either sell first, then short, or cover first, then buy. We only allow the first of one of these trades to occur during a single trading session. The second trade is added to the restricted list so that it may be performed during a later session. The two trades that involve side changes have been added to the `restricted` list.

```
> tl@restricted

    id orig target side shares   mv            reason
1  HAL    0    200    B    200 4000 Side change enter
2 YHOO    0     25    B     25  500 Side change enter
```

We have added the buy candidates for HAL and YHOO to the restricted data frame so that we do not accidentally enter a box position. The `reason` column explains why these candidates have been added to `restricted`. During this trading session we will attempt to exit the short positions for HAL and YHOO by covering these positions. In a subsequent trading session we will attempt to enter a long position by buying these equities.

5.3 Creating sorts and assigning them weights

Like in the previous example, we name the sorts and assign them weights by creating a list.

```
> sorts <- list(alpha = 1, ret.1.d = 1/2)
```

We assigned a weight of 1 to alpha and a weight of 0.5 to one-day return.

5.4 Passing additional information to `tradelist`

We must pass a data frame with columns for `id`, `price.usd`, `volume`, `alpha`, and `ret.1.d` in the call to `new`:

```
> sub.data

       id volume price.usd alpha ret.1.d
IBM   IBM   2100        10 -0.76  -0.003
GOOG GOOG   2200        20  1.10   0.010
GM     GM    100        30  1.00   0.100
SCHW SCHW   2500        50  1.20  -0.010
MSFT MSFT   2600        60  1.50  -0.020
AMD   AMD   3000       100 -0.94  -0.040
```

[14] Writing code so that we make a side change without creating a box position is hard. We will address this in future versions of the `portfolio` package

```
DELL DELL   3100        110 -0.15  -0.020
EBAY EBAY   3200        120 -0.32  -0.070
QCOM QCOM   3900        190 -0.36  -0.005
HPQ   HPQ   4000         15 -1.30  -0.002
HAL   HAL   4000         20  1.70   0.001
PAYX PAYX   4000         25  0.53  -0.001
APPL APPL   4000         30 -0.30  -0.090
TXN   TXN   4000         25 -0.50  -0.010
YHOO YHOO   4000         20  1.20  -0.002
```

Aside from having information about additional equities, this data frame does not differ greatly from the one we passed to new in section 3.3.

5.5 Calling `new`

Having gathered the components necessary to build a tradelist `tradelist`, we make a call to `new`:

```
> tl <- new("tradelist", orig = p.current, target = p.target,
+     chunk.usd = 2000, sorts = sorts, turnover = 36825, data = data)
```

We pass 8 arguments as parameters to the `new` method. The parameters are similar to those in section 3.3 with the exception of turnover which we have set to $36,825. The value of the candidate trades in this example is greater than the value of the candidate trades in the previous example so we must set `turnover` higher if we want to complete all of the candidate trades.

6 The `tradelist` algorithm, long-short

The way the `portfolio` package builds a long-short `tradelist` is similar to the way it builds a long-only `tradelist`. We will walk through the process of creating a long-short `tradelist` with `portfolio` and discuss the differences between creating long-only and long-short trade list.

6.1 Calculating ranks

We calculate the ranks for a long-short portfolio in much the same way we do so for a long-only portfolio. The main difference we must take into is the need to rank four types of trades with other trades of the same type. In previous examples we ranked buys and sells separately. Now we rank buys, sells, covers, and shorts separately.

6.1.1 Raw ranks with a long-short `tradelist`

As per our third simplifying assumption, we do not favour one type of trade over another type of trade. As a consequence, we split and rank the trades separately.

```
$B
       id orig target side shares   mv alpha rank
MSFT MSFT    0    100    B    100 6000   1.5    1
SCHW SCHW    0    100    B    100 5000   1.2    2
GOOG GOOG    0     50    B     50 1000   1.1    3
GM     GM  100    200    B    100 3000   1.0    4

$C
       id orig target side shares   mv alpha rank
HAL   HAL  -75      0    C     75 1500  1.70    3
YHOO YHOO  -10      0    C     10  200  1.20    7
PAYX PAYX -100    -50    C     50 1250  0.53   11

$S
       id orig target side shares    mv alpha rank
AMD   AMD  150    100    S     50 -5000 -0.94    1
QCOM QCOM   75     50    S     25 -4750 -0.36    2
DELL DELL   50      0    S     50 -5500 -0.15    3

$X
       id orig target side shares    mv alpha rank
HPQ   HPQ  -50   -100    X     50  -750  -1.3    1
TXN   TXN  -25    -50    X     25  -625  -0.5    2
APPL APPL    0    -75    X     75 -2250  -0.3    3
```

Like on page 24, the `$B` data frame shows the buys ranked with other buys and the `$S` data frame shows the sells ranked with other sells. The `$C` and `$X` data frames show covers and shorts ranked with other shorts.

6.1.2 Interleaving

The last step left us with 4 sets of ranks, one for each type of trade. Up to four trades will share each rank when we combine these data frames to form a list of overall rankings and the trades will be interleaved using groups of up to four.[15]

```
       orig target side shares    mv alpha rank
B.MSFT    0    100    B    100  6000  1.50    1
S.AMD   150    100    S     50 -5000 -0.94    1
X.HPQ   -50   -100    X     50  -750 -1.30    1
B.SCHW    0    100    B    100  5000  1.20    2
S.QCOM   75     50    S     25 -4750 -0.36    2
X.TXN   -25    -50    X     25  -625 -0.50    2
B.GOOG    0     50    B     50  1000  1.10    3
C.HAL   -75      0    C     75  1500  1.70    3
S.DELL   50      0    S     50 -5500 -0.15    3
X.APPL    0    -75    X     75 -2250 -0.30    3
B.GM    100    200    B    100  3000  1.00    4
C.YHOO  -10      0    C     10   200  1.20    7
C.PAYX -100    -50    C     50  1250  0.53   11
```

[15]Some of the groups may not include one trade of every type.

As per the third simplifying assumption, there is no natural way to choose between the best buy, sell, cover, or short. To deal with this ambiguity, we always break ties in rank between a buy, sell, cover, and short by assigning the buy the highest rank, the sell the second highest rank, the cover the third highest rank, and the short the worst rank:

```
     orig target side shares    mv alpha rank
MSFT    0    100    B    100  6000  1.50    1
AMD   150    100    S     50 -5000 -0.94    2
HAL   -75      0    C     75  1500  1.70    3
HPQ   -50   -100    X     50  -750 -1.30    4
SCHW    0    100    B    100  5000  1.20    5
QCOM   75     50    S     25 -4750 -0.36    6
YHOO  -10      0    C     10   200  1.20    7
TXN   -25    -50    X     25  -625 -0.50    8
GOOG    0     50    B     50  1000  1.10    9
DELL   50      0    S     50 -5500 -0.15   10
PAYX -100    -50    C     50  1250  0.53   11
APPL    0    -75    X     75 -2250 -0.30   12
GM    100    200    B    100  3000  1.00   13
```

Once again, each candidate has a unique rank and the rows appear in groups of buys, sells, covers, and shorts. The pattern repeats throughout he data frame because we have ties at every rank except for the last. There is no tie at the last rank because we have an odd number of candidates.

6.1.3 Weighted ranks

Having interleaved the separate rankings by type, we calculate weighted ranks.

```
       id orig target side shares    mv alpha rank
MSFT MSFT    0    100    B    100  6000  1.50    1
AMD   AMD  150    100    S     50 -5000 -0.94    2
HAL   HAL  -75      0    C     75  1500  1.70    3
HPQ   HPQ  -50   -100    X     50  -750 -1.30    4
SCHW SCHW    0    100    B    100  5000  1.20    5
QCOM QCOM   75     50    S     25 -4750 -0.36    6
YHOO YHOO  -10      0    C     10   200  1.20    7
TXN   TXN  -25    -50    X     25  -625 -0.50    8
GOOG GOOG    0     50    B     50  1000  1.10    9
DELL DELL   50      0    S     50 -5500 -0.15   10
PAYX PAYX -100    -50    C     50  1250  0.53   11
APPL APPL    0    -75    X     75 -2250 -0.30   12
GM     GM  100    200    B    100  3000  1.00   13
```

We double the one-day return ranks to reflect that one-day return is less important than alpha. (Recall that lesser ranks are better.)

```
> tl@rank.sorts[["ret.1.d"]]
```

```
        id orig target side shares    mv ret.1.d rank
GM      GM  100    200    B    100  3000   0.100    2
AMD    AMD  150    100    S     50 -5000  -0.040    4
HAL    HAL  -75      0    C     75  1500   0.001    6
APPL APPL    0    -75    X     75 -2250  -0.090    8
GOOG GOOG    0     50    B     50  1000   0.010   10
DELL DELL   50      0    S     50 -5500  -0.020   12
PAYX PAYX -100    -50    C     50  1250  -0.001   14
TXN    TXN  -25    -50    X     25  -625  -0.010   16
SCHW SCHW    0    100    B    100  5000  -0.010   18
QCOM QCOM   75     50    S     25 -4750  -0.005   20
YHOO YHOO  -10      0    C     10   200  -0.002   22
HPQ    HPQ  -50   -100    X     50  -750  -0.002   24
MSFT MSFT    0    100    B    100  6000  -0.020   26
```

We assign each candidate the best weighted rank from either sort. We combine the data frame of the candidates ranked by alpha with the data frame of the candidates ranked by one-day return:

```
     id orig target side shares    mv rank
1   AMD  150    100    S     50 -5000    2
2   AMD  150    100    S     50 -5000    4
3  APPL    0    -75    X     75 -2250   12
4  APPL    0    -75    X     75 -2250    8
5  DELL   50      0    S     50 -5500   10
6  DELL   50      0    S     50 -5500   12
7    GM  100    200    B    100  3000   13
8    GM  100    200    B    100  3000    2
9  GOOG    0     50    B     50  1000    9
10 GOOG    0     50    B     50  1000   10
11  HAL  -75      0    C     75  1500    3
12  HAL  -75      0    C     75  1500    6
13  HPQ  -50   -100    X     50  -750    4
14  HPQ  -50   -100    X     50  -750   24
15 MSFT    0    100    B    100  6000    1
16 MSFT    0    100    B    100  6000   26
17 PAYX -100    -50    C     50  1250   11
18 PAYX -100    -50    C     50  1250   14
19 QCOM   75     50    S     25 -4750    6
20 QCOM   75     50    S     25 -4750   20
21 SCHW    0    100    B    100  5000    5
22 SCHW    0    100    B    100  5000   18
23  TXN  -25    -50    X     25  -625    8
24  TXN  -25    -50    X     25  -625   16
25 YHOO  -10      0    C     10   200    7
26 YHOO  -10      0    C     10   200   22
```

To remove duplicates, we assign each candidate the best weighted rank associated with it by any sort.

```
     orig target side shares    mv
MSFT    0    100    B    100  6000
AMD   150    100    S     50 -5000
GM    100    200    B    100  3000
HAL   -75      0    C     75  1500
HPQ   -50   -100    X     50  -750
SCHW    0    100    B    100  5000
QCOM   75     50    S     25 -4750
YHOO  -10      0    C     10   200
APPL    0    -75    X     75 -2250
TXN   -25    -50    X     25  -625
GOOG    0     50    B     50  1000
DELL   50      0    S     50 -5500
PAYX -100    -50    C     50  1250
```

Once again we generate raw ranks:

```
     orig target side shares    mv rank
MSFT    0    100    B    100  6000  1.0
AMD   150    100    S     50 -5000  2.5
GM    100    200    B    100  3000  2.5
HAL   -75      0    C     75  1500  4.0
HPQ   -50   -100    X     50  -750  5.0
SCHW    0    100    B    100  5000  6.0
QCOM   75     50    S     25 -4750  7.0
YHOO  -10      0    C     10   200  8.0
APPL    0    -75    X     75 -2250  9.5
TXN   -25    -50    X     25  -625  9.5
GOOG    0     50    B     50  1000 11.0
DELL   50      0    S     50 -5500 12.0
PAYX -100    -50    C     50  1250 13.0
```

Having created weighted ranks, we prepare for the creation of synthetic ranks.

6.1.4 Mapping to the truncated normal distribution

We create synthetic ranks from by mapping the ranks to a truncated normal distribution. We scale buys and covers to the the 85^{th} percentile and above and sells and shorts to the 15^{th} percentile and below $((0, 0.15] \cup [0.85, 1))$.

```
     side alpha ret.1.d rank rank.ws
AMD     S -0.94  -0.040    2   0.037
HPQ     X -1.30  -0.002    4   0.043
QCOM    S -0.36  -0.005    6   0.075
APPL    X -0.30  -0.090    8   0.107
TXN     X -0.50  -0.010    8   0.107
DELL    S -0.15  -0.020   10   0.112
GOOG    B  1.10   0.010    9   0.880
PAYX    C  0.53  -0.001   11   0.887
```

```
SCHW    B  1.20  -0.010    5   0.910
YHOO    C  1.20  -0.002    7   0.925
GM      B  1.00   0.100    2   0.940
HAL     C  1.70   0.001    3   0.962
MSFT    B  1.50  -0.020    1   0.970
```

Finally, we map the values to the truncated normal distribution:

```
> tl.ranks

       id orig target side shares    mv alpha ret.1.d rank.t
AMD   AMD  150    100    S     50 -5000 -0.94  -0.040   -1.8
HPQ   HPQ  -50   -100    X     50  -750 -1.30  -0.002   -1.7
QCOM QCOM   75     50    S     25 -4750 -0.36  -0.005   -1.4
APPL APPL    0    -75    X     75 -2250 -0.30  -0.090   -1.2
TXN   TXN  -25    -50    X     25  -625 -0.50  -0.010   -1.2
DELL DELL   50      0    S     50 -5500 -0.15  -0.020   -1.2
GOOG GOOG    0     50    B     50  1000  1.10   0.010    1.2
PAYX PAYX -100    -50    C     50  1250  0.53  -0.001    1.2
SCHW SCHW    0    100    B    100  5000  1.20  -0.010    1.3
YHOO YHOO  -10      0    C     10   200  1.20  -0.002    1.4
GM     GM  100    200    B    100  3000  1.00   0.100    1.6
HAL   HAL  -75      0    C     75  1500  1.70   0.001    1.8
MSFT MSFT    0    100    B    100  6000  1.50  -0.020    1.9
```

6.2 Calculating chunks

Calculating chunks for a long-short portfolio functions in almost the same manner as it would for a long-only portfolio. We set the market value of each chunk to be 2,000 in the call to `new`.

```
> sub.chunks

       side rank.t chunk.shares chunk.mv tca.rank
AMD.1     S   -1.8           20    -2000     -1.8
AMD.2     S   -1.8           20    -2000     -1.8
AMD.3     S   -1.8           10    -1000     -1.8
APPL.1    X   -1.2           67    -2010     -1.2
APPL.2    X   -1.2            8     -240     -1.2
DELL.1    S   -1.2           18    -1980     -1.2
DELL.2    S   -1.2           18    -1980     -1.2
DELL.3    S   -1.2           14    -1540     -1.2
GM.1      B    1.6           67     2010      1.6
GM.2      B    1.6           33      990     -4.4
GOOG.1    B    1.2           50     1000      1.2
HAL.1     C    1.8           75     1500      1.8
HPQ.1     X   -1.7           50     -750     -1.7
MSFT.1    B    1.9           33     1980      1.9
MSFT.2    B    1.9           33     1980      1.9
MSFT.3    B    1.9           33     1980      1.9
```

```
MSFT.4    B    1.9            1      60      1.9
PAYX.1    C    1.2           50    1250      1.2
QCOM.1    S   -1.4           11   -2090     -1.4
QCOM.2    S   -1.4           11   -2090     -1.4
QCOM.3    S   -1.4            3    -570     -1.4
SCHW.1    B    1.3           40    2000      1.3
SCHW.2    B    1.3           40    2000      1.3
SCHW.3    B    1.3           20    1000      1.3
TXN.1     X   -1.2           25    -625     -1.2
YHOO.1    C    1.4           10     200      1.4
```

Aside from the addition of cover and short chunks, the chunk table should appear exactly as it does in section 4.3.

6.3 Calculating Swaps

Swaps work slightly differently with a long-short tradelist than with a long-only tradelist. In a long-only tradelist we only have to pair buys and sells, but in a long-short tradelist we have to pair buys, sells, shorts, and covers. The `calcSwaps` method accounts for this by matching trades within a side. We pair shorts with covers and buys with sells:

```
> swaps.sub

             side.enter tca.rank.enter side.exit tca.rank.exit rank.gain
MSFT.1,NA.0           B            1.9         S      -10000.0   10001.9
MSFT.2,NA.0           B            1.9         S      -10000.0   10001.9
MSFT.3,NA.0           B            1.9         S      -10000.0   10001.9
MSFT.4,NA.0           B            1.9         S      -10000.0   10001.9
HPQ.1,NA.0            X           -1.7         C       10000.0   10001.7
GM.1,NA.0             B            1.6         S      -10000.0   10001.6
SCHW.1,NA.0           B            1.3         S      -10000.0   10001.3
APPL.1,NA.0           X           -1.2         C       10000.0   10001.2
APPL.2,NA.0           X           -1.2         C       10000.0   10001.2
TXN.1,NA.0            X           -1.2         C       10000.0   10001.2
SCHW.2,AMD.1          B            1.3         S          -1.8       3.1
SCHW.3,AMD.2          B            1.3         S          -1.8       3.1
GOOG.1,AMD.3          B            1.2         S          -1.8       3.0
GM.2,QCOM.1           B           -4.4         S          -1.4      -3.0
NA.0,HAL.1            X        10000.0         C           1.8   -9998.2
NA.0,QCOM.2           B       -10000.0         S          -1.4   -9998.6
NA.0,QCOM.3           B       -10000.0         S          -1.4   -9998.6
NA.0,YHOO.1           X        10000.0         C           1.4   -9998.6
NA.0,DELL.1           B       -10000.0         S          -1.2   -9998.8
NA.0,DELL.2           B       -10000.0         S          -1.2   -9998.8
NA.0,DELL.3           B       -10000.0         S          -1.2   -9998.8
NA.0,PAYX.1           X        10000.0         C           1.2   -9998.8
```

In the `side.enter` column we list buys (`B`) and shorts (`X`) because the only way to enter a side is by initially buying or shorting a stock. Sells and covers move us closer to exiting the position which is why we put these trades in the `side.exit` column. Like in previous examples, the labels describe the swaps. The value to the left of the comma is the name of buy or short and the name to the right of the comma is the name of a sell or cover. The number following the period is the chunk number of the stock involved in the trade.

Dummy chunks work similarly for long-short portfolios as they do for long-only portfolios. The main difference is that we must create dummy shorts and covers to pair with real covers and shorts. We create 18 dummy chunks. The dummy chunks at the head of the swaps table exist because the current portfolio has a lesser market value than the target portfolio. To increase the market value of the current portfolio we want to make more buys and covers than sells. The dummy chunks at the tail of the table were created because we ran out of shorts and buys to match with real covers and sells. We assign this type of dummy trade a poor trade-cost adjusted rank.

6.4 The `calcSwapsActual` Method

The `calcSwapsActual` method works in almost exactly the same way as it does for a long-only tradelist.

```
> sub.swaps.actual <- tl@swaps.actual[, c("side.enter", "tca.rank.enter",
+     "side.exit", "tca.rank.exit", "rank.gain")]

> sub.swaps.actual
```

	side.enter	tca.rank.enter	side.exit	tca.rank.exit	rank.gain
MSFT.1,NA.0	B	1.9	S	-10000.0	10001.9
MSFT.2,NA.0	B	1.9	S	-10000.0	10001.9
MSFT.3,NA.0	B	1.9	S	-10000.0	10001.9
MSFT.4,NA.0	B	1.9	S	-10000.0	10001.9
HPQ.1,NA.0	X	-1.7	C	10000.0	10001.7
GM.1,NA.0	B	1.6	S	-10000.0	10001.6
SCHW.1,NA.0	B	1.3	S	-10000.0	10001.3
APPL.1,NA.0	X	-1.2	C	10000.0	10001.2
APPL.2,NA.0	X	-1.2	C	10000.0	10001.2
TXN.1,NA.0	X	-1.2	C	10000.0	10001.2
SCHW.2,AMD.1	B	1.3	S	-1.8	3.1
SCHW.3,AMD.2	B	1.3	S	-1.8	3.1
GOOG.1,AMD.3	B	1.2	S	-1.8	3.0
GM.2,QCOM.1	B	-4.4	S	-1.4	-3.0

We do not remove any swaps because we set the `turnover` equal to the unsigned market value of the candidate trades. If we decrease `turnover`, some of the swaps will be excised.

```
> tl@turnover <- nt - tl@chunk.usd
```

We set turnover to equal the turnover necessary to complete all of the candidate trades (`nt`), minus the maximum size of a chunk. This guarantees that we do not make trade the worst swap, in this case NA.0,PAYX.1. By lowering `turnover` we caused the worst ranked swap to be removed.

```
> sub.swaps.actual

             side.enter tca.rank.enter side.exit tca.rank.exit rank.gain
MSFT.1,NA.0           B            1.9         S      -10000.0   10001.9
MSFT.2,NA.0           B            1.9         S      -10000.0   10001.9
MSFT.3,NA.0           B            1.9         S      -10000.0   10001.9
MSFT.4,NA.0           B            1.9         S      -10000.0   10001.9
HPQ.1,NA.0            X           -1.7         C       10000.0   10001.7
GM.1,NA.0             B            1.6         S      -10000.0   10001.6
SCHW.1,NA.0           B            1.3         S      -10000.0   10001.3
APPL.1,NA.0           X           -1.2         C       10000.0   10001.2
APPL.2,NA.0           X           -1.2         C       10000.0   10001.2
TXN.1,NA.0            X           -1.2         C       10000.0   10001.2
SCHW.2,AMD.1          B            1.3         S          -1.8       3.1
SCHW.3,AMD.2          B            1.3         S          -1.8       3.1
GOOG.1,AMD.3          B            1.2         S          -1.8       3.0
GM.2,QCOM.1           B           -4.4         S          -1.4      -3.0
```

6.5 Calculating actual chunks

The `calcchunksActual` method works similarly to the way it does for a long-only tradelist:

```
> sub.chunks.actual

       side alpha ret.1.d rank.t tca.rank chunk.shares chunk.mv chunk
MSFT.1    B  1.50  -0.020    1.9      1.9           33     1980     1
MSFT.2    B  1.50  -0.020    1.9      1.9           33     1980     2
MSFT.3    B  1.50  -0.020    1.9      1.9           33     1980     3
MSFT.4    B  1.50  -0.020    1.9      1.9            1       60     4
HPQ.1     X -1.30  -0.002   -1.7     -1.7           50     -750     1
GM.1      B  1.00   0.100    1.6      1.6           67     2010     1
SCHW.1    B  1.20  -0.010    1.3      1.3           40     2000     1
APPL.1    X -0.30  -0.090   -1.2     -1.2           67    -2010     1
APPL.2    X -0.30  -0.090   -1.2     -1.2            8     -240     2
TXN.1     X -0.50  -0.010   -1.2     -1.2           25     -625     1
SCHW.2    B  1.20  -0.010    1.3      1.3           40     2000     2
SCHW.3    B  1.20  -0.010    1.3      1.3           20     1000     3
GOOG.1    B  1.10   0.010    1.2      1.2           50     1000     1
GM.2      B  1.00   0.100    1.6     -4.4           33      990     2
AMD.1     S -0.94  -0.040   -1.8     -1.8           20    -2000     1
AMD.2     S -0.94  -0.040   -1.8     -1.8           20    -2000     2
AMD.3     S -0.94  -0.040   -1.8     -1.8           10    -1000     3
QCOM.1    S -0.36  -0.005   -1.4     -1.4           11    -2090     1
       dummy.quality
MSFT.1          <NA>
MSFT.2          <NA>
```

```
MSFT.3          <NA>
MSFT.4          <NA>
HPQ.1           <NA>
GM.1            <NA>
SCHW.1          <NA>
APPL.1          <NA>
APPL.2          <NA>
TXN.1           <NA>
SCHW.2          <NA>
SCHW.3          <NA>
GOOG.1          <NA>
GM.2            <NA>
AMD.1           <NA>
AMD.2           <NA>
AMD.3           <NA>
QCOM.1          <NA>
```

We have changed the swaps back into chunks. The additional work for a long-short portfolio involves converting buy/sell and short/cover swaps into chunks instead of just dealing with buy/sell chunks.

6.6 The `calcActual` Method

The `calcActual` method works almost exactly the same way it does for a long-only tradelist:

```
> tl@actual

       id side shares    mv alpha ret.1.d rank.t
AMD   AMD    S     50 -5000 -0.94  -0.040   -1.8
APPL APPL    X     75 -2250 -0.30  -0.090   -1.2
GM     GM    B    100  3000  1.00   0.100    1.6
GOOG GOOG    B     50  1000  1.10   0.010    1.2
HPQ   HPQ    X     50  -750 -1.30  -0.002   -1.7
MSFT MSFT    B    100  6000  1.50  -0.020    1.9
QCOM QCOM    S     11 -2090 -0.36  -0.005   -1.4
SCHW SCHW    B    100  5000  1.20  -0.010    1.3
TXN   TXN    X     25  -625 -0.50  -0.010   -1.2
```

We “roll-up” all the chunks into single orders.

7 Conclusion

With intelligently defined sorts, the `portfolio` package is a powerful tool for managing equity portfolios. Nonetheless, the `tradelist` code could stand for improvement in certain areas, particularly the area of trade-cost adjustment. The current method of using discrete and static boundaries for determining trade-adjusted rank should be replaced by a trade-cost adjustment function. Nonetheless, we believe that our package makes the difficult problem of trading a little bit easier.