A Volume-Weighted Average Paper

Allison Bishop[*]

1 Introduction

Very few things about stock market prices are straightforward. Most people know they can vary quite quickly as a function of time. But this is just the first of many complications. You can't just ask: "What is the price for Apple right now?", because the answer can depend on if you're buying or selling, how many shares you want to buy/sell, where you want to trade, and even how you ask. If you want to trade a significant amount of stock, you may obtain more favorable prices by splitting your large order into many smaller orders, spread out over time and space.

It is the task of algorithms to decide how to distribute orders over time and space to try to achieve "good" pricing for large amounts of trading activity. But how do we define the goal "good" pricing? Naturally, we want to buy low and sell high, but low and high are comparative terms. Low and high compared to what? If we take two different algorithms and use them to perform two different sets of trades, it's very hard to tell if the different prices achieved reflect a meaningful difference in algorithm performance, as the market conditions may vary widely between the two data sets.

There are several approaches to this problem. Elsewhere in our research, we chip away at the approach of characterizing market conditions by coarse features and attempting to flesh out patterns in how prices respond to such features. But here, we'll take another common path: targeting a volume-weighted average price, a.k.a. VWAP.

The spirit of a VWAP algo is perhaps best described as an "I'll have what he's having" approach. As a metric, VWAP is motivated by the following reasoning: if it's hard to mitigate the "noise" of the market that impacts the prices of trade, then perhaps we should compare those prices to something that is subject to the exact *same* noise. The something is the volume-weighted average price obtained across all trades in a particular stock for the same time period we are trading that stock. For example, let's suppose we are buying shares of MSFT throughout a trading day, and we ultimately purchase 10,000 shares for a total of 2,000,000 dollars. Then the volume-weighted average price for our shares is $\frac{2000000}{10000} = \$200$. On the same day, let's suppose that a total of 300,000 shares of MSFT were traded, and a total of $\$59,850,000$ dollars changed hands as a result. Then the volume-weighted average price across the market for that day is \$199.50. In this case, we view \$199.50 as the benchmark for our trading, since it averages prices obtained by others trading the same stock during the same time period as us. In this case, we paid a little more than the benchmark.

As a metric for algorithmic performance, comparing our VWAP to the general market VWAP has some pros and cons. For one thing, it does considerably remove noise from our measurements. However, it also introduces a circularity: we are measuring ourselves against a benchmark that we also influence. If we are trading heavily, our own activity will impact the prices across the market. This creates a blind spot for price impact to hide: whatever impact makes it into the benchmark will not show up in the difference between our performance and the benchmark. For this reason, we continue to explore alternative metrics elsewhere (see https://www.prooftrading.com/docs/distilled-impact.pdf).

Another pro of VWAP is that it suggests a natural secondary goal: distributing your trading volume throughout the day in a way that is similar to how overall trading volume is distributed. This goal is related to a well-defined prediction problem. In the next sections, we will formulate this prediction problem, introduce our preliminary approach, and summarize our results and the first version of our VWAP algorithm.

[*]Proof Trading, allison@prooftrading.com

2 Defining the Prediction Problem

Suppose we want to buy X shares of a particular stock over the course of a day, and we want to match the volume-weighted average price as closely as possible. For now, we'll imagine we are going to start trading at the beginning of the trading day (including the opening auction) and stop trading at the close (putting any remaining volume into the closing auction). Later we'll discuss cases where we are seeking to trade over a smaller time interval that may start later or end earlier.

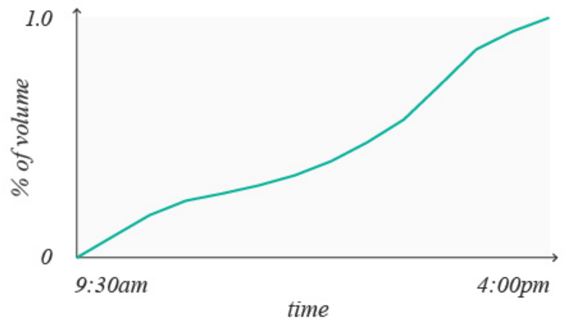
A typical approach to this is to try to trade roughly in line with the "volume curve," meaning that if 10% of the day's volume trades within in a certain time window, then we try to buy about $\frac{X}{10}$ shares within that time window. We should note up front that this may not be necessary: someone could offer a crossing mechanism, for example, that matches shares to be traded and waits for the true VWAP price of the day to be determined before pricing the trades.

But for now, we'll stick to thinking about the volume curve. We start to see the challenge when we zoom in on the phrase "10% of the day's volume..." in our above example. This is something that will not be known until the end of the day. If we divide the trading day from 9:30 am to 4 pm into disjoint 10-minute buckets of time, for example, by 9:40 am we will know how much volume traded in the first bucket, but we will not yet know what percentage of the day's total this represents. But we are already facing the decision of how many shares do we want to try to buy over the next bucket. Clearly we cannot wait until the end of the day to decide that.

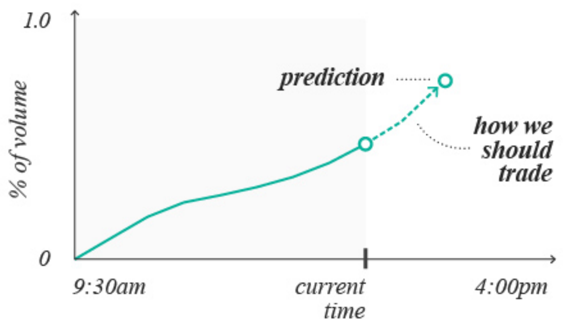
There is a mismatch here between the units of our task (buy X shares), and the units of volume we are trying to match (trade Y% of volume in this time interval). If we wanted to say buy X% of the volume over the course of the day but we didn't care how many absolute shares this represented, then we could easily track our progress against this metric in real time and try to slow down or catch up as desired. If we wanted to say buy X shares out of the next Y traded shares, we could similarly track our progress in real time. But wanting to buy an absolute number of shares and distribute it proportionately makes is less obvious how we might use real time data to adjust our actions. Perhaps this is why many VWAP algorithms largely ignore real time data and rely upon "volume curve" predictions informed by recent completed trading days.

A typical default is to look at, say, the last 20 completed trading days, and compute the average percentage of volume falling into each time bucket for each symbol. We can equivalently view this as giving us a prediction for the cumulative percentage of volume that we expect to have traded by the end of each time interval. If we treat this prediction as truth, we know what we should do at the beginning of each interval. If we have traded Y% of our X shares so far, and we expect Z% of the day's volume to have traded by the end of the interval, than we should try to trade an additional $(Z - Y)$% of our X shares in this interval.

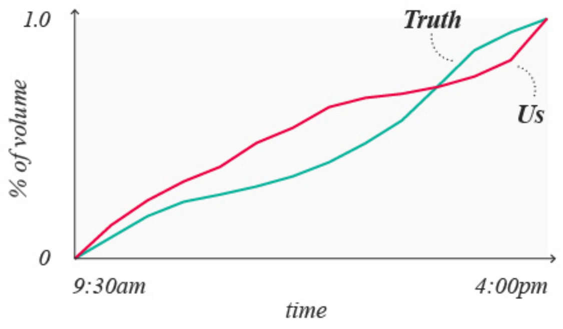
For visualization purposes, let's make the (false) simplifying assumption that over each interval, the volume trades at a steady rate. In fancy terms, this means we are approximating the cumulative volume percentage by a piece-wise linear function of time. In other words, we are imagining that the percentage of volume traded so far as a function of time behaves something like this:



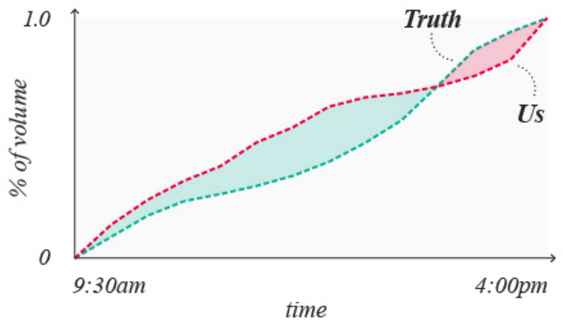
So if we have been right on target so far, and we correctly predict where we want to be by the end of the next interval, we know what we need to do:



But of course, reality is unlikely to be that cooperative. And solely using pre-computed averages leaves us no mechanism for adjusting when we are off. After the day is completed, we'll know the true picture, and we can evaluate our performance. We can plot the true curve of cumulative relative volume versus time, which we'll call $Truth(t)$, and the curve of our trading, which we'll call $Us(t)$:

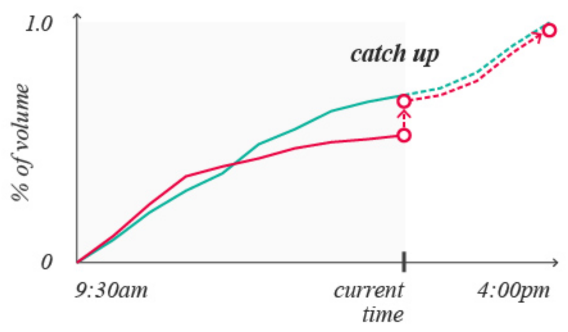


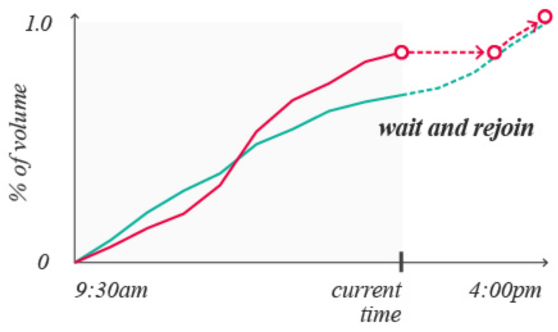
There are many reasonable choices for how we might quantify our error, but we will choose the total area between the curves.



In math-speak, that is the integral of $|Truth(t) - Us(t)|$. We like this metric for several reasons. It is easy to visualize, and intuitively punishes us equally for being ahead or behind. It also reflects our desire to correct errors as quickly as possible. For example, if we are currently behind and we are seeking to minimize $|Truth(t) - Us(t)|$, we should want to catch up as quickly as possible, rather than say gradually making up our deficit over the remaining time. This is desirable behavior because the price of later trading is likely to deviate further and further away from the current price, so making up our deficient later is likely to result in greater deviation from the VWAP price than making it up now. Setting a goal of minimizing the integral of $|Truth(t) - US(t)|$ over time is one way of incorporating this. (There may be an exception if we are so far behind that our quickly catching up would be unusual and likely to cause inordinate price impact. But generally, we believe that catching up small amounts sooner rather than later is a sound strategy under this metric.)

So how might we reduce our error according to this metric? Let's suppose that we could use real time data to make higher quality predictions of what percentage of the day's volume has traded so far as well as what percentage will have traded by the end of the next time interval. Our updated predictions might suggest that we are ahead or behind our target. If our goal is to minimize the area between the $Truth(t)$ and $Us(t)$ functions over time and we think we are behind, we should attempt to immediately catch up and then follow a linear path to the next predicted value. If we think we are ahead, we should wait for our prediction of Truth(t) to catch up to us and then follow a linear path to the next predicted value:





What we need to implement this strategy is: 1) a prediction of the current cumulative volume percentage and 2) a prediction of the next cumulative volume percentage. Clearly, the typical 20-day rolling averages could be used as a default for these, but next we'll see how to use real time data to improve the quality of these predictions.

3 Predicting cumulative relative volume with real time data

For a given symbol over a given trading day, we'll let $R(t)$ denote the cumulative relative volume traded up to time t. In other words:

$R(t) :=$ (Volume traded from 9:30 am up until time t) / (Volume traded from 9:30 am through 4:10 pm).

To accommodate the fact that the closing auction sometimes occurs slightly after 4 pm, we have used the cutoff time of 4:10 pm instead of 4 pm sharp. We'll let $V(t)$ denote the volume traded up to time t, so we can rewrite this as:

$$R(t) := V(t)/V(4:10pm).$$

We note that at time t, $V(t)$ is known, but $R(t)$ is not yet known.

We'll be breaking the day into 10-minute intervals, so we are focusing for now on the discrete sequence of values:

$$R(9:30am), R(9:40am), R(9:50am), \ldots, R(3:50pm), R(4:00pm), R(4:10pm).$$

Note: it is a delicate issue how best to handle the open and close. Likely these should be treated individually and not mixed in with the rest of the trading day. For now, we'll abuse notation a little and think of $R(9:30am)$ as indicating the relative volume of the opening auction, $R(4:00pm)$ as indicating the relative volume of the opening auction and regular trading day exclusive of the closing auction, and $R(4:10pm)$ as including the closing auction.

Let's use t_1, t_2,... etc. to denote our discrete sequence of times. At time t_i, we want to make a prediction for both $R(t_i)$ and $R(t_{i+1})$. We emphasize that $R(t_i)$ is not known at time t_i, as only the numerator of $R(t_i)$ is known at time t_i, not the denominator.

We'll first assemble some pieces of historical information that may be highly relevant to predicting $R(t_i)$ and $R(t_{i+1})$ on a given day.

$$AVG_R(t_i) := \text{ a rolling 20-day average of } R(t_i)$$

$$AVG_R(t_{i+1}) := \text{ a rolling 20-day average of } R(t_{i+1})$$

$$ADV := AVG_V(4:10pm) = \text{ a rolling 20-day average of daily volume}$$

In order to start with a minimalist set of features, we'll use the values of ADV, and $V(t_i)$ to compute a single feature with an intuitive interpretation:

$$A(t_i) := V(t_i)/ADV$$

The numerator here is the actual volume that has occurred. Dividing by the average daily volume makes this a rough indicator of how far off we might be from where we expected to be at this point.

We next investigate whether considering the value of $A(t_i)$ as well as the value of $AVG_R(t_i)$ allows us to form a better estimate of $R(t_i)$ than using $AVG_R(t_i)$ alone. Similarly, we'll investigate if considering the value of $A(t_i)$ as well as the value of $AVG_R(t_{i+1})$ allows us to form a better estimate of $R(t_{i+1})$ than using $AVG_R(t_{i+1})$ alone.

What we'll do is group observations of our training data by their values of $AVG_R(t_i)$ and $A(t_i)$, rounded to the nearest two decimal points and one decimal point respectively. In each group, we'll compute the average value of $R(t_i)$ (weighting all observations equally within each symbol, but averaging across symbols weighted by notional value). We'll also do the analogous computation of the average value of $R(t_{i+1})$ for groups determined by their rounded values of $AVG_R(t_{i+1})$ and $A(t_i)$. This gives us two look-up tables: conditioned on the values of $AVG_R(t_i)$ and $A(t_i)$, we can look up a prediction for $R(t_i)$, and conditioned on the values of $AVG_R(t_{i+1})$ and $A(t_i)$, we can look up a prediction for $R(t_{i+1})$. [Technical aside: we omit from our tables any variable combinations that have a sample size $<= 100$. In these cases, the 20-day averages are used as defaults.]

By simple inspection of these tables, we see that the predictions do vary considerably from the $AVG_R(t_i)$ and $AVG_R(t_{i+1})$ values as the $A(t_i)$ values vary, suggesting that the $A(t_i)$ variable is indeed meaningful for predicting $R(t_i)$ and $R(t_{i_1})$.

We can compare these new look-up table predictions against the default of using $AVG_R(t_i)$ and $AVG_R(t_{i+1})$ on fresh test data. We trained the tables on data from January through March of 2019, and tested them on data from April through June 2019. In the test, we computed the average sum of squared errors in the default estimates for $R(t_i)$ and $R(t_{i+1})$ for each symbol, and then we averaged these over symbols weighting proportionally to the notional value traded in each symbol.

The (notional value-weighted) average sum of squared errors in the default estimates for $R(t_i)$ and $R(t_{i+1})$ was 0.0135, while the analogous value for our table look-up estimates was 0.0106. This represents a meaningful improvement.

To see if this held up for more tumultuous times, we also trained tables on data from January through March of 2020, and tested them on data from April through June 2020. The average sum of squared errors in the default estimates for $R_(t_i)$ and $R_(t_{i+1})$ was still 0.0122, while the analogous value for our table look-up estimates was 0.0103. (Note that we are squaring errors so that overestimates and underestimates are both penalized, but it's important to keep in mind that squaring numbers less than 1 makes them smaller. So the average individual error in magnitude here is more like $\approx \sqrt{\frac{0.01}{2}} = 0.07$. So we're saying that our individual estimates of cumulative volume percentages are typically off by about 7% with this technique, compared to about 8% for the baseline of 20-day averages.

The numbers for 2019 vs. 2020 were similar enough that we might wonder: how much does it matter when we train our table? Maybe the relationship between our variables is relatively stable over time. So we ran the same test on 2020 data (April - June again), but with the tables fit from 2019 data. The (NV-weighted) average sum of squared errors this time was still 0.0122 for the baseline (this is identical because it's unaffected by the tables) and became 0.0101 for the estimates assisted by our table look-ups. This is some evidence that the relationships between the variables are fairly stable over time, and may not benefit from frequent re-fitting.

It should be noted that the "20-day" quantities above were computed with time periods lasting 20 calendar days, not trading days. Given that trading does not occur on the weekend, approximately 30 calendar days would be needed to cover 20 trading days. We redid the approach above using 30 calendar days instead, training the tables of data from Jan - March 2019 and testing on data from April - June 2019. The NV-weighted average sum of squared errors this time was 0.0133 for the baseline and became 0.0106 when using the trained lookup tables. Using those same tables and testing on data from April - June 2020 yielded 0.0121 for the baseline and 0.0102 when using the lookup tables. Thus, there doesn't seem to be much difference between using 20 or 30 calendar days for the rolling averages. [We should note an annoying detail of our analysis setup. The "N-day" averages for the first N days in our input to a query are actually less than N-day averages: they are averages from the start of the input to the current day. This "wrong" initial condition affects the first 30 days in our input when we increase N to 30. Obviously fixing the analysis setup would be preferable to leaving it, but it's too annoying to be worth it at the moment.]

We note that the $V(t_i)'s$, $R(t_i)$'s, and the ADV are the only variables that need to be referenced for each symbol, which makes the resulting algorithm relatively easy to implement.

4 Extension to partial trading days

Let's now consider extending the reasoning above to cases where we have an intraday order with a start time that is past 9:30 am, and/or an end time that is before 4 pm. To make things concrete, let's imagine the order arrives at 1 pm and should finish trading by 2 pm, and it is an order to buy X shares. At 1 pm when the order starts, we have an estimate (based on historical data as well as real-time trade data so far) of what percentage of the total day's volume has already traded as of 1 pm. We also have an estimate for how much of the day's volume will have traded by 1:10 pm. But to decide how many of our X shares we should try to buy between 1 pm and 1:10 pm, we need to know one more thing: what percentage of the day's volume do we expect to have traded by our end time of 2 pm?

This reveals two new variables that were held constant when were only talking about full trading days: the percentage of volume traded before we started was fixed at 0, and the percentage of volume traded by our end time was fixed at 1. While these percentages at particular times were held constant, our estimates of percentages at other times could change and could become inconsistent as new data arrives. For example, we might estimate at 10 am that 10% of the day's volume has already traded by 10 am. But by 11 am based on updated real-time data, we might estimate that only 8% of the day's volume has already traded by 11 am.

Before, this wasn't a problem. We would just make our new decisions based on our new estimates and try to catch up or slow down as indicated by the newer data. But now, our estimates for the order start time and end time need to be referenced throughout. If we think, for instance, that 20% of the day's volume traded before our start time, that 60% of the day's volume will trade before our end time, that 30% of the day's volume has traded by the current moment, and that 40% will have traded by the end of the next 10 minutes, and we have already traded Y out of X shares, what should we do?

First we want to examine $\frac{Y}{X}$ and compare it to $\frac{30-20}{60-20} = \frac{1}{4}$. If $\frac{Y}{X}$ is greater than this value, we want to slow down. If it is less, we want to catch up. Let's say its less and we immediately buy Z shares so that $\frac{Y+Z}{X}$ is approximately $\frac{1}{4}$. Next, let Z' denote the number of shares such that $\frac{Y+Z+Z'}{X}$ is approximately $\frac{40-20}{60-20} = \frac{1}{2}$. Then Z' is how many shares we want to buy spaced out of the next 10 minutes. Employing this logic with the exact approach we described in the prior sections runs into a problem. We described a way to make new estimates of $R(t_i)$ and $R(t_{i+1})$ at time t based on up-to-date data, but we did not describe a way to also estimate $R(start)$ and $R(end)$. In fact, since the order start and end times are arbitrary, we would need to have a general method for re-estimating the entire function R over all values of t at any point in time, based on the real-time data so far.

Doing this well is a task we leave for future research. For now, we'll do something simpler: we'll fix our estimates of $R(start)$ and $R(end)$ at the time of the order start. These estimates will come from the default 20-day average. Once these are fixed, our later estimates of $R(t)$ for times t between the start and end time may end up violating basic sanity conditions. For instance, we could have a situation where our current estimate for $R(t)$ is less than our fixed estimate for $R(start)$, or where our current estimate for $R(t)$ is greater than our fixed estimate for $R(end)$. To enforce sanity, we'll replace any estimated value r for $R(t)$ with the maximum of r and $R(start)$ or with the minimum of r and $R(end)$ as necessary. We note that $R(end) \geq R(start)$ is guaranteed to hold when both values are estimated from 20-day averages on the same data set.

There is another minor detail we have glossed over. Technically, we have only described how to derive estimates for $R(t)$ at times t that fall on 10-minute boundaries, but an order might arrive at a time like 9:42 am. For this, we can linearly interpolate our estimates for 9:40 am and 9:50 am to arrive at an estimate for 9:42 am.

Obviously, there are a lot of things about this approach that can be improved and further explored. We expect to continue our research on predicting volume curves and matching VWAP prices as closely as possible, and to iteratively improve our VWAP algo over time. In particular, it would be desirable to make our methods less reliant on discretized intervals of time and react to real-time data in a more streaming fashion. One natural way to do this is to fit a continuous function to the prediction data, rather than stopping at an empirical aggregation.