Rejecting the Black Box: an Inside Look at the Design of Proof Trading's New Algorithm

Allison Bishop[*]

1 Prologue: Why should you publish an algo design?

Computer science as a practice requires you to define precisely what you mean. You cannot tell a computer: "trade stocks for me, and give me best execution!" You must patiently and painstakingly instruct it to copy some portions of bits into other portions of bits, shuffle some third set of bits around, read in some other bits from somewhere else, and so on. Well, usually not actually you, but someone. Well, actually lots of someones. Someones who designed the operating system you're working on, someones who designed the programming language you're using, someones who designed the network protocols your trading system uses to communicate with other people's systems, and so on.

These details are fundamentally knowable in nature, but not in scale by individual human beings. As computer systems evolve, we layer abstractions between ourselves and the lowest level operations of bits. We delegate to hierarchies of teams, tools, and vendors. We allow knowledge to pool into silos of narrow specialization, because otherwise, we could not keep up and get anything done.

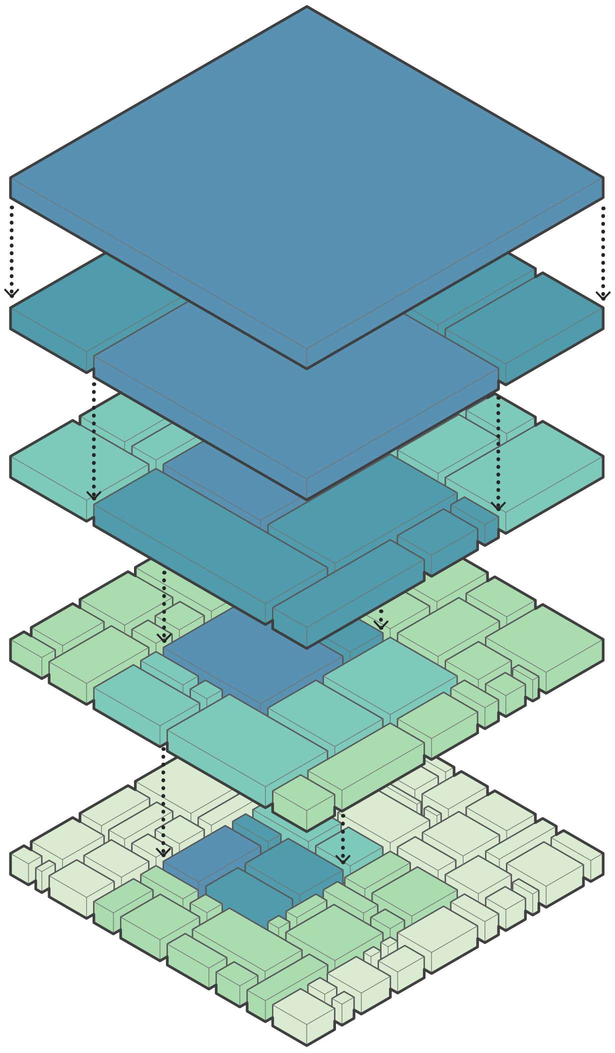
The notion of an "algorithm" sits both atop this hierarchy and outside it. If you consult a computer science textbook, the definition of algorithm will likely use words like "procedure" or "recipe" that are not intrinsically tied to the realm of computers. It's the instructions you give for accomplishing a task, it might say, or a sequence of steps to be followed. The language becomes awkward and vague, but not because the concept is new. Rather because the concept is old. So old, that we probably learned it before we gave it a name. We learned algorithms for tying our shoes, for adding two integers, for brushing our teeth. We learned these things as answers to the question of "how," and we learned that answers could have varying degrees of specificity. At first we needed very specific instructions, but as we learned and got older, we could follow higher level instructions, subsuming the lowest details as familiar, predictable pieces that did not need to be said explicitly anymore. We also learned

[*]allison@prooftrading.com

to subsume certain tools as given, and restrain ourselves from falling too deeply down every rabbit hole of "why."

This subsumption has obvious benefits, and less obvious costs. Sometimes we feel the costs when we try to teach someone else something we "know" but find ourselves unable to explain. Sometimes we feel the costs when something changes, and we don't know how to adapt our procedures effectively. In some cases, the lower level knowledge that informed our higher level understanding has evaporated from our minds, leaving only a derived residue that is brittle in its relation to context, and perhaps invisibly so. How long might it take us to recognize when we are operating on assumptions that are no longer true?

Layering of Abstraction & Subsumption of Details



There is something more that can be lost in layers of abstraction if we are not careful: the value of forcing ourselves to be explicit. Anyone who has ever taught a young child or programmed a computer is well aware of the phenomenon - we think we know what we mean, until we see our own words parroted back at us in a literal translation with horrifying consequences. Sometimes this is born of our failure to properly define a sub-concept we are referencing ("ok Tommy, I admit I did not really mean it when I said you could color 'anywhere' that wasn't on the wall"), and sometimes it is born of our failure to anticipate the circumstances under which our instructions will be applied ("oops! I thought I issued that delete instruction in the subfolder containing those old files, not the main folder containing the new ones!") In navigating our lives and careers, we humans are quick to grasp that there is safety in distance from such specific commands. "But honey, I clearly told the babysitter to keep him out of trouble!" and "But I only instructed the intern to delete the *unnecessary* files!" We often pretend that all we gain from our distance is convenience and efficiency, but often we are seeking less accountability as well.

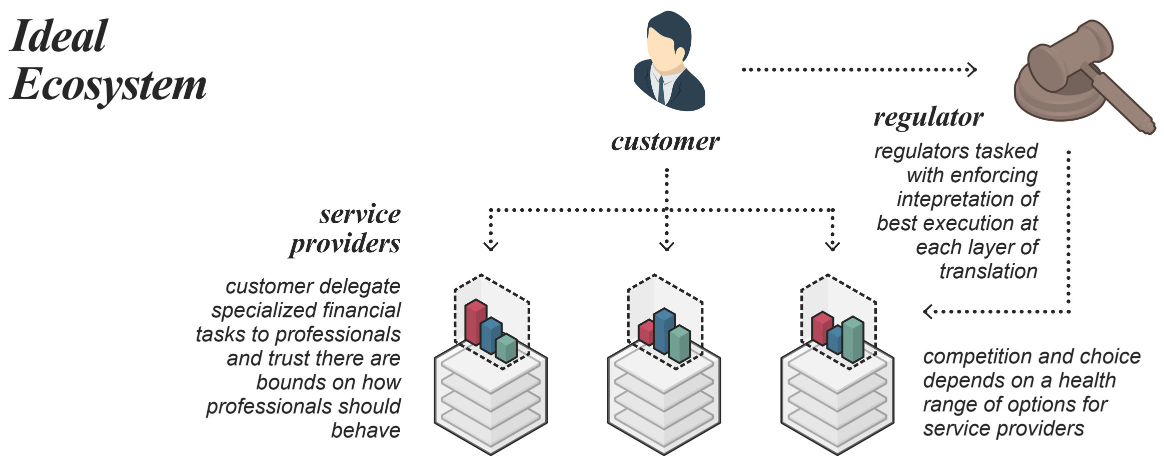
Specificity is hard and risky work. But *somebody* does it, even if we don't. Between every layer, somebody has to translate higher level instructions into lower level ones until we get all the way down to the shuffling of bits or the changing of diapers. The supposedly shared context that "goes without saying" is meant to make this seamless, but this foundation can crumble rather easily and dramatically at times.

In the domain of algorithmic trading, shared context may be under quite a bit of strain. It is a nearly comical game of telephone that begins when someone puts money into a retirement account. "Give me growth or something," the future retiree says. "Give them growth or something," the fund manager tells his subordinates. "Give me exposure to these three factors that roughly mean growth or something," the next person says. Some number of iterations later, someone says "Give me 100,000 shares of MSFT." By this point, an important translation has occurred. The original customer goal, to the extent that it was ever formulated in the first place, has been pooled with other customers' goals, melded through other intermediaries' interpretations and blended with their own separate goals, and one or more specific orders to buy/sell stocks has emerged from this process. The benefit to the end customer so far is supposed to be twofold. One benefit is the supposed superiority of this chain of experts as compared to the customer's own haphazard guess at translating high level goals into concrete orders. The other benefit is diversification: since stocks are ultimately bought and sold in indivisible units called shares, it isn't possible for a single customer with a more limited amount of money and time to purchase and actively maintain the same diversity of assets that a fund manager can purchase and maintain with the pool of all their customers' money. One can certainly debate the true extent of these benefits, and compare them to more automated solutions like indexing and various artificial notions of fractional shares. But it's at least somewhat clear what problems these services are supposed to be solving. It would be quite unreasonable and inefficient for each individual retiree to build up the body of knowledge required to passably translate "growth or something" into a suitable portfolio of financial assets and continually re-balance it over time.

Conversely, there is danger lurking in the end customer's ability to vaguely say "growth or something." The portfolio that emerges may not serve the customers needs well at all. Or the customer may have wildly unrealistic expectations of performance or risk. Such things may be the result of honest miscommunication, deliberate subterfuge, or misaligned incentives (or all of the above).

This is not a problem that is particular to finance, or to computer driven systems. It is a fundamental tension inherent in all task delegation: when you save yourself the work of making your instructions explicit all the way down to the lowest detail, you introduce opportunities for an agent who does not fully understand or share your goals to deviate from what you want them to do in certain circumstances if you had taken the time to fully understand the details.

A popular mechanism for navigating this is competition and choice. Agents will compete for your business, and you can reward the ones who do a good job by continuing to use them, and punish the ones who do a bad job by terminating their services. This mechanism works well when a few conditions are satisfied: 1. there is a healthy range of options for service providers, 2. distinguishing between a good job and a bad job is something that can be done in a reasonable amount of time, and is a much easier problem than doing a good job in the first place.


Ideal Ecosystem
customer
regulator
regulators tasked with enforcing intepretation of best execution at each layer of translation
service providers
customer delegate specialized financial tasks to professionals and trust there are bounds on how professionals should behave
competition and choice depends on a health range of options for service providers

In the case of a customer contributing to a retirement account, that second condition is problematic. The funds are supposed to perform well over a long term time horizon, and judging them on a short term basis is likely to be dominated by market noise and yield little insight. As a result, there are vast sub-industries of finance organized around addressing these tensions, and vast regulatory regimes in place to try to protect end customers and enforce at least a reasonable zone of interpretation at each layer of translation. Pretty much everyone agrees that this is necessary. Customers need to be able to delegate specialized financial tasks to professionals and trust there are bounds on how professionals should behave. Competition alone is not a sufficient mechanism, as customers aren't readily equipped to evaluate sophisticated products without putting in an unreasonable amount of work. As new layers emerge and old layers evolve, it's a constantly moving and delicate dance.

And in fact, we've only started. The game of telephone keeps going. The next person says "Give me 25,000 shares of MSFT today and probably 25,000 more tomorrow, we'll see how it goes." The next person says "allocate today's 25,000 share order to one of our brokers and ensure best execution."

Let's pause again for a moment. Something weird happened there. Things were still getting more concrete, but then a new source of vagueness slipped in: the notion of best execution. It sounds pretty innocent: who wouldn't want "best" execution? But what does it actually mean?

If you consult FINRA rule 5310 on Best Execution and Interpositioning, you find that a broker must "use reasonable diligence to ascertain the best market for the subject security and buy or sell in such market so that the resultant price to the customer is as favorable as possible under prevailing market conditions." This language rules out some obviously bad and lazy things, like routing all customer orders to a particular dark pool without ever comparing the results to other possibilities. But it leaves a lot of wiggle room. There are two gaping holes in this guidance: one is lurking in the phrase "under prevailing market conditions." Since the execution of trades is an interactive process between the many brokers submitting orders and the multiple venues matching orders, the timing of trades is highly variable. Timing of individual trades is not completely within a broker's control (they can't control when willing counter-parties arrive), but it is influenced heavily by the choices the broker makes in how to distribute a large order into many small orders over time and over trading venues, and how the broker communicates orders to trading venues (use of order types and order parameters). Since "prevailing market conditions" change rapidly in time, the influence a broker exerts over timing is also an influence on the "prevailing market conditions" under which the trade will be executing. In this way, brokers affect both the grade and the grading rubric for best execution at the same time.

The second gaping hole is that the best execution guidance doesn't really grapple with the nature of large orders, which are unlikely to be traded in their entirety at once. When a broker designs an algorithm to break up a large order into smaller pieces and seek to trade the pieces gradually throughout the trading day, does the best execution responsibility apply to just the pieces individually or to the large order as a whole? Clearly in spirit, it should apply to the large order as a whole. But what does a "price to the customer ... as favorable as possible under prevailing market conditions" even mean when you are looking at several individual prices over the course of a day where market conditions were changing dynamically? How can you know what would have been possible if the order had been chopped up in a different way? What is the space of "reasonable" alternatives that one should compare to and how does one do so while general market noise is likely to drown out small differences in outcomes due to the broker's behavior? And if you give up on this harder problem and just evaluate each small trade in its temporally local context where things are clearer, surely you might be blind to important failures to choose the "best" local times and order sizes.

So what happens after this troublesome notion of "best execution" is introduced into the game of telephone? It's not too hard to guess. It gets parroted down the line for a bit, then disappears into the black box of a secret "algo". When the telephone game turns around and each person reports back to their superior, the "best execution" straw man re-emerges at the same point and gets passed back up. "Buy me 25,000 shares of MSFT today using your VWAP algorithm that provides best execution," the next person tells the broker. The algorithm makes its choices of how to slice up the 25,000 shares, and spits outs a dynamic sequence of much more specific commands: "place a midpoint peg buy order for MSFT on NASDAQ at 10:01:02 am for 100 shares," the algo says. These commands get transmitted through multiple network layers (and often multiple vendors and intermediaries) and finally land at a trading venue, where perhaps they result in a trade. Their fate gets passed back up to the algo, which may adjust its state and issue new orders. The algo passes its results back to the broker who is running it. The broker periodically runs some paltry and horribly noisy tests on these results to make sure they seem reasonable. Then the broker passes back up to the next person: "here's your volume-weighted average price, achieved with best execution." This continues to percolate up the levels to the originator of the 100,000 share mandate, who ultimately receives their 100,000 shares of MSFT, their bill, and an assertion of "best execution." Here the "best execution" notion evaporates again, and the message morphs back into "here's your growth or something" to the retiree, who can check the behavior of their account and try to keep it consistent with their goals at various time horizons.

There are two questions that arise when we critically examine this workflow. First: are algo designers really the best people to translate this vague notion of "best execution" into specific sequences of orders in a dynamic, distributed market? Second: how does the commonly black-box nature of algos contribute positively and negatively to the overall process? How much visibility should an algo provide, and to whom?

We firmly believe the answer to the first question is yes. The problem of algorithm design for electronic trading is a complex scientific problem. It involves the delicate dynamics of distributed systems, the complex economics of continuous trading and batched auctions, the fraught task of modeling market forces as randomized processes, and the herculean statistical challenge of evaluating alternative choices in a meaningful way when the degrees of freedom combined with the inherent variance conspire to overwhelm the sample size of a single firm's trading activity. This is a problem that deserves to be tackled by scientists. Retirees, regulators, and even financial professionals with other specialities should not be expected to solve these kind of problems for themselves and then tie the algo designers' hands.

But is competition a sufficient mechanism to ensure that algo designers will do a "good" job on behalf of the end clients? While there is a healthy number of agency brokers and algo products available for trading US equities, it is not at all clear that those who choose between the products can evaluate their effectiveness with a sufficient amount of accuracy within a reasonable use of energy and time.

Let's do a thought experiment (informed by real market data) to help gauge the extent of the challenge to evaluation. Each trading day, the official opening price of a stock is set through an auction at 9:30 am, and the price fluctuates continuously throughout the day until the official closing price is set in an auction at 4:00 pm[1].

If we look at the sequence of prices obtained for trades of a given stock on a given trading day, there is a significant amount of fluctuation. To get a rough sense of how much, we can look at the relative change from the opening price to the closing price. If we let O_p denote the opening price and C_p denote the closing price, then this quantity is defined as:

$$D_p := \frac{C_p - O_p}{O_p}.$$

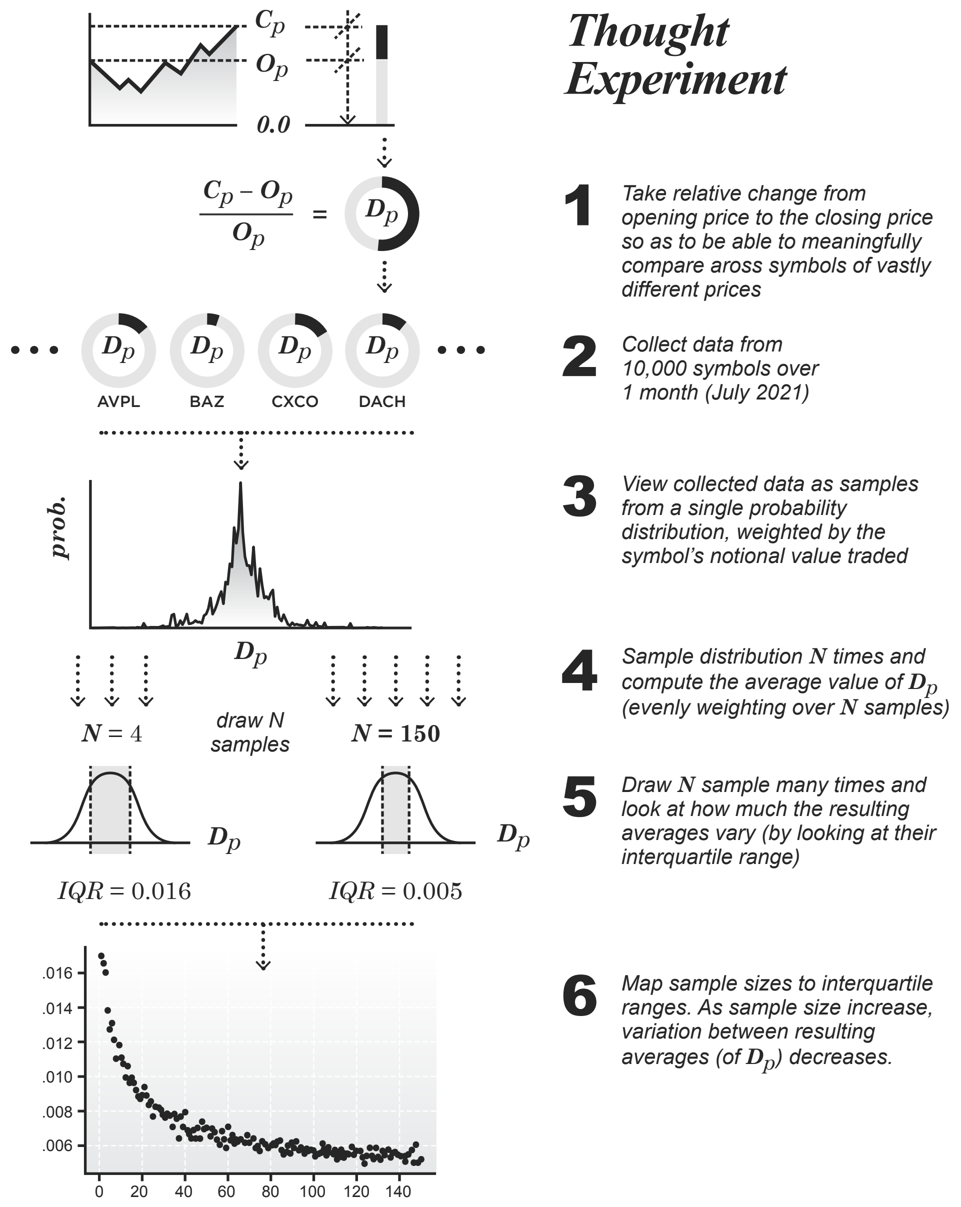
We have made this relative to the opening price so that it is meaningful to compare this quantity across different stocks that have vastly different prices. On a given day, we will have over $10,000$ values of D_p, one for each symbol. If we collect these D_p values over symbols and over trading days, we can view them as samples from a single probability distribution, weighted by notional value. In other words, we can build up an empirical estimate of the underlying distribution by placing probability mass on each observed D_p value that is proportional to the notional value traded in that symbol on that day.

We did this for all symbols and all trading days over the month of July 2021. Once we have this probability distribution in hand, we can sample it any N times and compute the average value of D_p (evenly weighting over our N samples). In some sense, N represents the number of orders we might use in a sample to try to measure an algo's performance. This is not a perfect analogy, as each sample here is drawn according to notional value, and real institutional trading flow will probably be distributed differently over symbols than the general notional value distribution over the market. Nonetheless, this should give us some intuition for how much variance there might be in our performance metrics. We can do many experiments of drawing N samples, and look how much the resulting averages vary. In particular, we'll look at the interquartile range of our resulting averages, which is the difference between the 75th and 25th percentiles.

For each N from 1 to 150, we did 1000 experiments, and took the difference between the 750th and 250th resulting values after sorting. Below is a graph of those interquartile ranges

[1]This is already ignoring some nuance. Sometimes opening or closing auctions are late or don't happen, and the notion of "price" is multi-faceted. The stock "price" might refer to the prices of actual trades (which happen at discrete times and for varying amounts of shares), or quoted prices from willing buyers or sellers, which are distinct from each other and last for certain windows or time before being updated, traded, or canceled. And some trading occurs in pre-market and post-market sessions, before 9:30 am and after 4 pm, respectively.

as N, the sample size for each experiment, grows from 1 to 150:



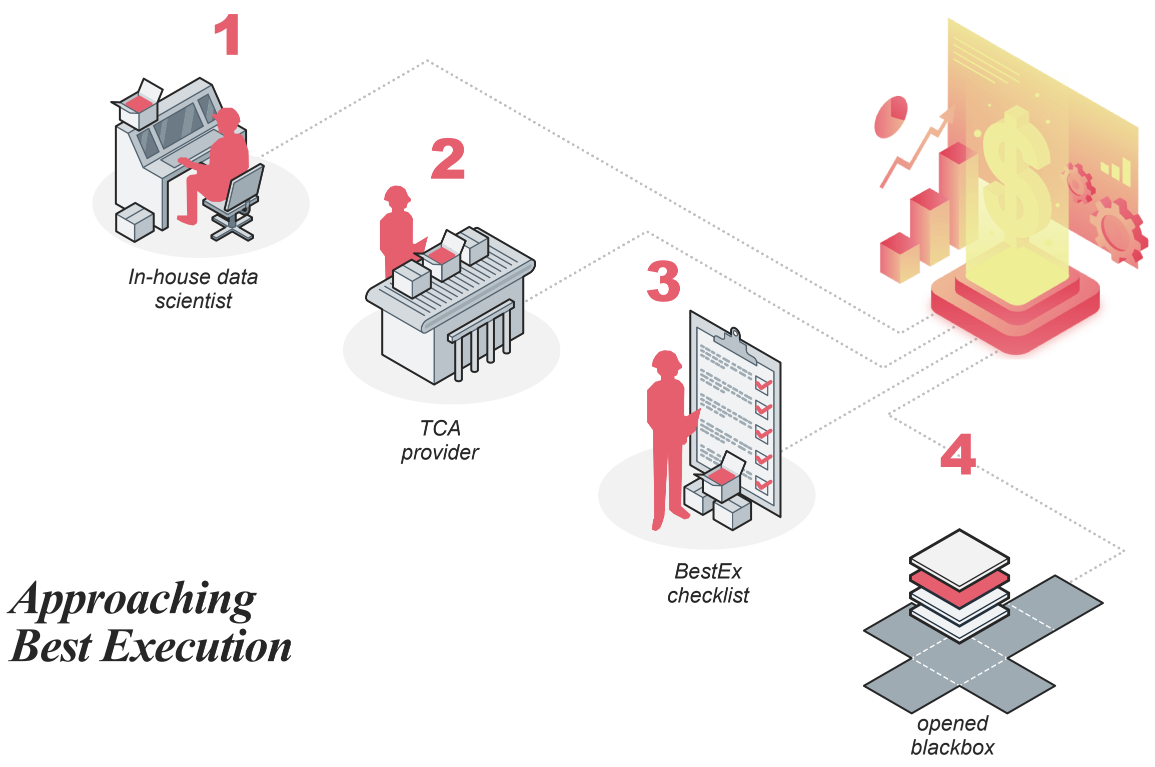
Thought Experiment
C_p
O_p
0.0
$\frac{C_p - O_p}{O_p} = D_p$
1 Take relative change from opening price to the closing price so as to be able to meaningfully compare aross symbols of vastly different prices
D_p AVPL
D_p BAZ
D_p CXCO
D_p DACH
2 Collect data from 10,000 symbols over 1 month (July 2021)
prob.
D_p
3 View collected data as samples from a single probability distribution, weighted by the symbol's notional value traded
$N = 4$
draw N samples
$N = 150$
4 Sample distribution N times and compute the average value of D_p (evenly weighting over N samples)
D_p
$IQR = 0.016$
$IQR = 0.005$
5 Draw N sample many times and look at how much the resulting averages vary (by looking at their interquartile range)
.016
.014
.012
.010
.008
.006
0
20
40
60
80
100
120
140
6 Map sample sizes to interquartile ranges. As sample size increase, variation between resulting averages (of D_p) decreases.

Unsurprisingly, as the sample size of each experiment increases, the variation between the resulting averages decreases. At sample sizes near 150, the interquartile ranges are a bit greater than 0.005 wide, and the improvement in precision as a function of growing sample size has slowed.

So what does this mean? For one thing, it suggests that it's quite difficult to see meaningful performance differences between algos on metrics like slippage vs. arrival at these sample sizes, unless the performance differences are considerably larger than 50 bps. For A-B testing different algos, especially when we are limited to the flow of a single client over a period of a month or two, this is pretty sobering news. Without further correction and careful normalization (a fraught process itself), extraneous market forces are likely to pull the results around noisily enough to obscure even meaningful and consistent differences in algo performance. In fact, *measuring* algo performance may be as hard (or harder!) as designing algos in the first place. And the people with the skill set to tackle this hard problem? You guessed it - the same people with the skill set to design algos.

This creates a very uncomfortable position for the first person in the game of telephone who is directed to ensure "best execution." To truly embody the full spirit of this directive seems to be a full time job equivalent to designing algos, and that's supposed to be the thing that's delegated to the black box! What to do?

There are a few common approaches to try to wriggle out of this conundrum. One is to hire in-house scientists to grade the outputs of the algo black boxes. Another is to outsource this job to an independent third party (a TCA provider). A third option is to hollow out the vague directive of "best execution" and replace it with a checklist that boils down to something more like "not obviously terrible execution."



All of these options have major drawbacks. The use of in-house scientists is likely the best, but it is also costly, and if everyone did it there would be some comical effects: the overall population of quantitative scientists would spend much more resources on grading algos than designing algos, and that feels like an inefficient state for the market as a whole. Also, finding, training, and crucially *listening to* good data scientists is a much harder problem than proliferating data science boot camps would like you to believe. The outsourced TCA provider at least allows a single set of scientists to serve as algo evaluators for a large population of firms who need to evaluate algos, but the incentives are a little weird. While it is true that the third party evaluators should have no incentive to cherry-pick the stats in favor of any particular algo, they also have no real strong incentive to do a good job, nor any clear mandate on what a good job is. Human beings are creatures of inertia, afterall, and most of what clients want from TCA providers is a stamp of approval that what they are doing already is basically ok. Providing that stamp is much easier to do if one combines TCA with the third option: fixing a minimally defensible definition of "best execution" rather than a formulating a more satisfying but complex one and having to teach your clients that this is what they *should* want.

There is a fourth option that, as far as we know, has not really been tried before: open the black box. What if we didn't limit ourselves to grading algos solely on noisy performance metrics? Naturally we'd always want to measure those to learn anything we reasonably can, but what if we could also cut through the noise and examine the raw source: the algo designs themselves, and the processes that drove their development?

As an illustrative comparison, consider the task of deciding where to send your child to school. You might look at test scores for each contender and these are likely to reveal any huge differences, but small differences are unlikely to be particularly meaningful. You could stop there and say, "I'll send my child to this school that has reasonable test scores," but wouldn't you also want to know *how* the various schools approach their mission of education? Ideally you would want to visit the various schools, you would want to talk to the teachers. You would want to know what they think is important, what they think is unimportant. You would want to gauge how much thought they have put into their approach, and how aligned their values are with your values. You would want to see what's underneath the test scores. Why settle for a noisy outcome evaluation when you can also directly assess the mechanisms that drive the outcomes?

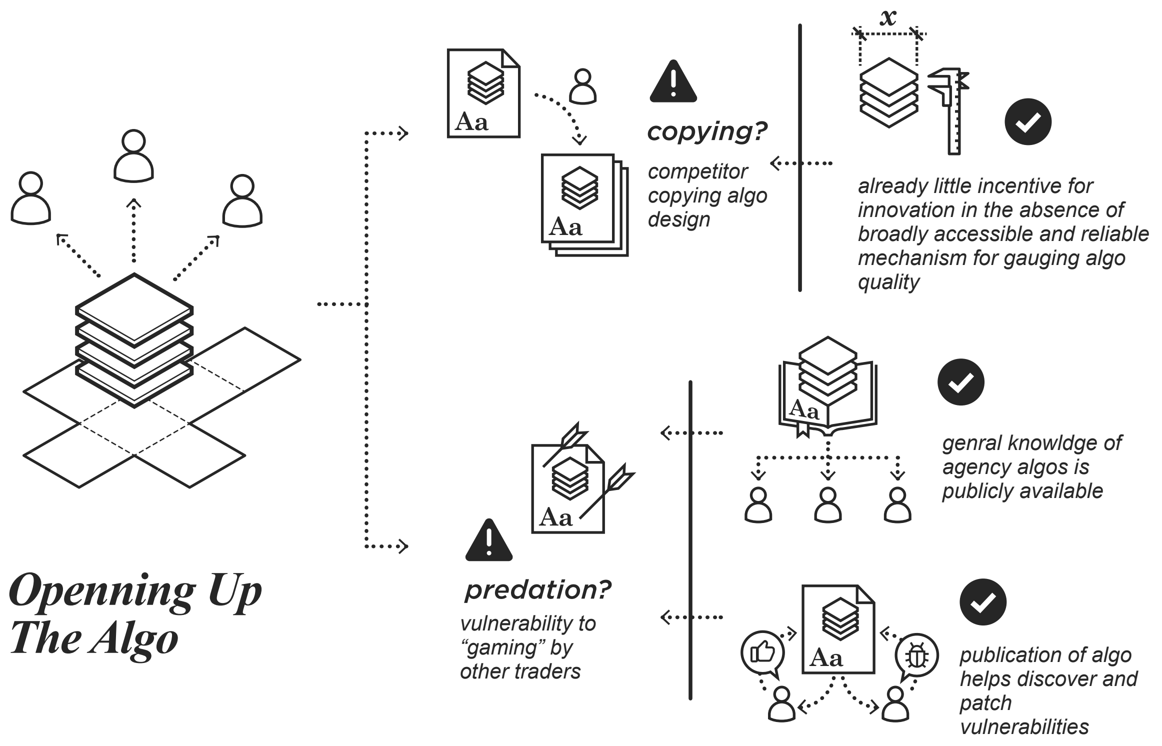
It's true that human beings are not great at this. Our minds are subconsciously manipulated by many heuristic habits that bias our assessments. We believe far too strongly in first impressions. We give undue weight to recent experiences, we are prone to falsely equate what is familiar with what is desirable, etc. But it is a fantasy to think that "data" on its own can save us from these cognitive traps. Our minds are instinctual and persuasive storytellers, and we can spin a story around ambiguous data about as easily as we can in a vacuum. For this reason, we should not wholly replace the challenging process of subjective assessment with blind reliance on noisy metrics.

So what can a non-algo designer reasonably hope to extract from a disclosed algo design and an account of the research behind it? Hopefully at least a few things like: 1. a sense of what kind of scientific processes the designers employ, 2. an understanding of what goals the designers are prioritizing, 3. an awareness of the assumptions the designers are making, 4. a rough idea of the extent and level of competency of the research, and last but certainly not least: 5. an opportunity to collaborate more directly with the designers and aim their expertise more effectively at achieving particular goals.

Many would argue that the potential downsides of publicly disclosing algo designs outweigh the value of these kind of assessments and collaborations. The most common arguments given are 1. competitors can copy a disclosed algo design and 2. a disclosed algo design is more vulnerable to being "gamed" by other traders. In a direct sense, 1. is only a problem for the company providing the algo, not its customers. But indirectly, one might worry that copied designs will remove incentive for innovation. This concern is circular though, because the incentive for innovation is already weak in the absence of a broadly accessible and reliable mechanism for gauging algo quality.

Concern 2. above is directly relevant to the clients of an algo, and it is certainly worth taking seriously. Let's think about what it means for a design to become "gameable" due to public information about its development. The process would be: someone reads the newly public information, combines it with their own current knowledge, and comes up with an idea to behave differently in their own trading algorithms and potentially improve their own outcomes at the expense of the disclosed algo's customers. If this worked to a significant extent, it would have to either 1. essentially work against a large portion of agency algos or 2. involve a step of approximately identifying the disclosed algo (or something very like it) in the wild. We must keep in mind here that someone looking to exploit the disclosed algorithm will not know what side/stocks/amounts the algo is actively trading on any given day. This is private information that comes from the customers and is never disclosed.

If 1. is true, then the role of the algo design disclosures is likely coincidental. General knowledge about how agency algos typically work is available already, and the set of people across the industry who have direct experience working on or around agency algos is not small. If 2. is true, then the disclosed algo is doing something unusually noticeable, either in its general behavior or in its response to conditions that a would-be gamer manufactures. In this case, the design has a problem, and should be fixed. Not disclosing the design is a flimsy protection in this case. Whatever the noticeable and exploitable behavior is, it could also be discovered by someone searching for such a thing, even if that person didn't know ahead of time what exactly to look for. In our age of big data and fast technology, such an unguided search could take longer than a targeted one, but perhaps not that much longer.



x
copying?
competitor copying algo design
already little incentive for innovation in the absence of broadly accessible and reliable mechanism for gauging algo quality
genral knowldge of agency algos is publicly available
Opennning Up The Algo
predation?
vulnerability to "gaming" by other traders
publication of algo helps discover and patch vulnerabilities

If we assume that any serious exploit will be eventually discovered, then our goal should be to discover and patch it ourselves as quickly as possible. Publication of our algo design and research supports this goal, as it enables us to collaborate with others more freely, and to vet our design through a larger audience. This is the same approach that is used to produce strong encryption algorithms like AES (the advanced encryption standard) that we all rely on to secure our sensitive communications (e.g. using our credit cards for online transactions). The design of AES is fully public and has been subject to extensive public vetting from the cryptologic research community for decades. The sensitive information encrypted via AES is protected by secret key values which are unknown to would-be attackers, but everything about *how* the secret keys and the sensitive information is combined to form an inscrutable ciphertext is known.

We believe that everyone up the telephone chain from the algo black box would be better served by a translucent box - and that's why we commit to publishing the research that goes into the design of our algorithms, as well the design of the ways we evaluate performance. The rest of this paper will detail the process we used to design the scheduling component of our new trading algorithm, as well as the twists and bumps we encountered along the way.

Our design process is heavily driven by the desire to learn as much as we can from historical market data, which is available to us in a quantity that is orders of magnitude greater than our own live trading data will represent for a long time. By evaluating potential features of the design extensively on historical market data, rather than solely relying on noisy A-B tests in live trading, we can improve our design much more quickly and more robustly. Historical market data can tell us a lot about how the market is likely to react to common situations. Once we've developed such information, we can start to model how the market may react to potential choices that our algo may make. Finally, we can derive the choices that our algo will make by comparing the modeled market reactions to the available choices and choosing the path that our modeling predicts will be most favorable for our execution goal.

Our execution goal is formulated mathematically in a later section, but its simplified version is basically: "don't shit where you eat." In this context, it means: try not to pay more as a buyer because *you* have pushed the price up. This is close in spirit to minimizing "impact," but lots of people use that term without converging on a single mathematical meaning, so we want to be a little more specific. Its most direct meaning is also not quite what we care about - we may not care if our activity moved the price *after* we were mostly done trading. We care more about how our activity so far drives up the prices we will incur in our remaining activity. In other words, we care about prices over time proportionately to how much we trade at those times. So we will seek to model how our behavior affects prices at a forward marching sequence of times, and we will define a cost function that ultimately calculates: according to our model of market reactions, what's the additional premium we expect to pay as a buyer due to our own actions driving up the stock over our sequence of trades? Naturally, we design the algo to choose the actions that minimize our estimate of this cost function, subject to accomplishing the desired total amount of trading over the time period.

There is one big question this paper will not answer. It is a question we get asked a lot: by would-be investors, colleagues in the industry, potential clients, and even our families on occasion. "Just me give me ballpark," they start, "how much money do you think you can ultimately save your clients?" We sigh. It's obvious why everyone wants an answer. It would certainly make our lives easier if we just gave an answer. We could hedge it in all the typical ways: "This is just a projection but..." and"If you assume ..." But frankly, there's currently no scientifically responsible way to answer this question. We could point to the paper "Trading Costs" by Frazzini, Israel, and Moskowitz [2], which estimates AQR's average market impact over many years of trading data to be roughly 9 bps, with about 1.26 bps of that being "transitory" impact that reverses soon after AQR's trading activity completes. This seems to suggest at least that trading costs overall do represent a significant term in the overall costs of institutional investing. But how much of this term is inherent, and how much is attributable to differences between algos? We don't know. It's very hard to know! We will work diligently to combat the confusion of market noise in our own iterative research process, and we are optimistic that we will be able to achieve reasonable and compelling estimates of how much better each version of our algo is compared to the last. But we won't

[2]available at `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3229719`

be able to compare ourselves to other algos because, well, *cough*, those algos are hiding in their black boxes.

So we can't tell you how much money we would save for a potential client. Because we don't know how much money their current brokers are really saving/costing them. And they don't know either. And isn't *that* unsettling? If we know that this cost term may be big enough to matter, and we know that we don't know how to control it with a noisy competition between black boxes, isn't that a good enough reason to force the boxes open?

It's great that a person doesn't need to become an expert in portfolio management, algorithmic trading, settlement and clearing, market microstructure, and more in order to accomplish an investment goal of "growth or something." And it's true that most investors have no interest in going down the telephone chain and understanding how their goal gets translated into something more concrete at each layer. But shouldn't that translation be knowable in principle? Shouldn't *someone* be empowered to check that each lower layer is doing a reasonable job of embodying the higher layer's wishes?

We think so. But we don't expect that other algo designers will shed their black boxes anytime soon. We'll just be here in the meantime, tinkering away in public, and happy to hear your thoughts on what we're building.

2 Introduction: A High-Level View of Proof's Trading Algorithm

The work of designing a trading algorithm is basically the work of filling in the empty space between a set of low level trading tools and a high level objective. This work has several stages, not necessarily performed in the following order:

- Determining the set of trading tools: deciding which venue connections and order types to use
- Developing a framework for assembling the tools into an algorithm
- Fleshing out the high level objective into more detailed metrics
- Developing a framework for comparing different versions of the algorithm according to the desired metrics

An intuitively appealing and perhaps common approach is: assemble as many trading tools as possible, throw them into an algorithm that mostly follows the "gut" of experienced traders, decide on a standard TCA metric like slippage vs. vwap or slippage vs. arrival, and then A-B test different versions of the algo on real trades, choosing whichever one performs best on the TCA metric (probably defaulting to no changes if the performance difference is too small or seems unreliable).

There are several things about this approach that are unsatisfying, however. The data science part here - the A-B testing - is being asked to work in highly sub-optimal conditions. The final outcomes on price are very noisy, and the sample size of live trading performed by the algo is relatively small. This makes setting the criteria for adopting changes to the algo very fraught. If the criteria set a higher bar for robustness and clarity of results, precious few proposed changes will clear the bar. But if the criteria set a lower bar, noise is likely to usher in a bloat of questionable "upgrades."

Another unsatisfying aspect of this approach is its distribution of labor between human intuition and quantitative science. Both are certainly needed! However, it's important to use humans for what humans are good at, and use data science for what data science is good at. Not only do we have data science working in sub-optimal conditions here, but we also have humans working in sub-optimal conditions. The search space of "ways that one could assemble underlying tools into a full algo" is vast and high-dimensional. This is not a great environment for human intuition to operate blindly in. An overreaction to this would be to fully replace human intuition with a fancy machine learning algorithm to quickly churn through the vast, high-dimensional search space. This would be a fine idea, except for the still unsolved noisy evaluation problem. If there isn't an extremely reliable way to compare alternatives, searching through too many possibilities in an automated fashion is a recipe for drowning in coincidences.

What we need instead is a way for humans and machines to work together to combat the noisy evaluation problem on a smaller, well-structured search space, with much more data at their disposal. Machine learning works best when it is employed to find simple structures on large data sets. So to set ourselves up for success, we'll look for ways to use human intuition to intelligently limit and structure the search space, as well as ways to enlarge the available data.

The first structure we impose is a modular and hierarchical one. Our algos consist of several layers of logic, each with distinct responsibilities. The highest layer decides how much of the total order to delegate to each of our underlying strategies, each of which represents a distinct execution goal. The decisions at this layer are controlled by parameters set by the client entering the order, as well as by our research which informs limits we place on how much volume a given strategy can or should successfully handle.

So far, we have designed three underlying strategies. One that we refer to as "VWAP" is intended to minimize slippage to the market's volume-weighted average price over the lifetime of the order. Another is a "liquidity seeker" whose goal is to find relatively big blocks. The third is an "impact minimizer" whose goal is to trade somewhat steadily while attempting to minimize the price impact of that trading activity.

Within each strategy, there is a "scheduler" layer whose job it is to decide how much to try to trade over medium-term timescales, which are roughly five to twenty minutes long. These decisions will be made with the context of how much has traded so far, how much of order's lifetime is left, as well as historical information like volume curves, and real-time information derived from market data or from the outcomes of recent actions taken by the algo. We can think of a scheduler as an oracle who answers questions like, "given what I know now and what's left to trade in my order, how much should I trade over the next X minutes?" Naturally, the answer to this question should depend on the execution goal, e.g. whether your goal is to get close to VWAP or to minimize price impact, etc. This is why each strategy has its own kind of scheduler.

Below the scheduler layer is a "tactical" layer whose job it is to decide *how* to trade the amount that the scheduler has prescribed for the next time interval, where time intervals are of randomized lengths typically within the range of five to twenty minutes. This is the layer that interacts with venues, decides when to post and when to take, which order types to use, etc.

In structural terms, we feel our intuition about market microstructure is very well-tuned. Accumulated across the Proof team, we have several decades of experience focusing on market microstructure in US equities. For the low level tactics of our algos, we are comfortable starting with a curated mix of passive and aggressive tools that we believe should serve our purposes well, including tools we ourselves heavily contributed to at IEX. We do expect to continually re-evaluate and improve upon these choices over time.

We have focused our research efforts so far at Proof on the layer where we expect our intuition is weaker and the room for improvement from a scientific approach may be considerable: the scheduler layer. The research behind our initial VWAP scheduler design is already published in a separate white paper (available at https://www.prooftrading.com/docs/vwap.pdf). In this paper, we present the research behind our initial design for the impact minimizing scheduler.

Having narrowed our goal for now to the design of a scheduler whose mandate is to "minimize impact" conditioned on completing a certain total amount of volume, our next task is to flesh out what we mean by "impact" and how we intend to go about measuring and minimizing it. Intuitively, we would like to say that impact represents how much a price moves *due to our trading activity.* But this is not something we can directly measure, as we don't know what would have happened *without* our trading activity. However, we can use various normalization techniques to try to reduce the confounding influence of wider market forces. We'll discuss such techniques more extensively in the next section.

Our basic notion of a price movement will be a relative one, and will be localized to ten minute time intervals at a time. We'll view the regular trading day in a given symbol as divided into 39 ten minute intervals, the first being from 9:30 am to 9:40 am, and the last being from 3:50 pm to 4 pm. For each of these intervals, we'll consider the relative price movement as the ratio of last trade price occurring in that interval over the last trade price occurring in the previous interval. (For the first interval, the denominator will be the first trade price occurring in that interval.) LP_i will represent the last trade price occurring in interval i, and the ratios we'll be interested in are of the form $\frac{LP_i}{LP_{i-1}}$. The evolution of the price over the course of the day can then be tracked by successively multiplying these ratios. It can be more convenient, however, to express these ratios as exponentials so that this price evolution process corresponds to addition in the exponent. In other words, for each interval i, we define the value Δ_i such that:

$$\frac{LP_i}{LP_{i-1}} = e^{\Delta_i}.$$

(i.e. Δ_i is the natural logarithm of the price ratio). In this notation, we can express the price at the end of i intervals the initial price multiplied by a single exponential term where the exponent is a sum of the first i values of Δ:

$$LP_i = e^{\sum_{j<=i} \Delta_j} FP,$$

where FP denotes the first price of the day (the opening price). We note that price movements that revert back to the starting price correspond to sequences of Δ_i values that sum to zero.

Having defined the Δ_i values that we use as measures of "impact" (subject to additional normalization techniques), our next task is to consider how we describe and study the relationship between the actions our algo will take and the times series of Δ_i values. We could try to study this solely using data collected from our own trading activity, where we clearly know what actions we took. However, even with normalization techniques that reduce the noise of general market movements on the Δ_i values for a given stock, it is very difficult to detect and model meaningful relationships between various quoting and trading behaviors and the time series of Δ_i values. This is a strong motivation for bringing in historical market data as resource, rather than relying solely on the small sample sizes of our own trading.

Leveraging historical market data requires us to define a translation between our own actions and features of trading and quoting activity that we can identify and study in the historical market data that we have. This is a bit challenging, as historical market data does not link or identify parties. We have price, size, and timing information about individual quote and trade events in historical market data, but no information about the underlying parties. Analogously, when our algo is taking actions in the market, our counter-parties will not have information that identifies us (at least in a direct sense. It is certainly part of our design challenge to avoid being identified indirectly by taking actions that are too distinctive or identifiable, etc.)

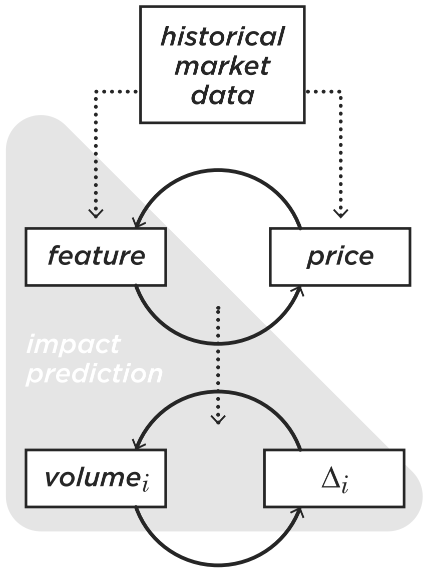
For each time interval and symbol in our historical market data, we can compute the value of Δ representing the exponent of the relative price change, and we can also compute various features of the trading and quoting behavior in that same interval, or in the interval(s) preceding it. (Ultimately, we choose to consider the behavior in the current interval and the one interval preceding it, so that we can model reversion effects from one interval to the next.) For example, we can compute things like: the total volume traded in the interval, the percentage of that volume that traded at the prevailing NBB/NBO/midpoint, the average size quoted at the NBB and NBO, etc. Overall, the set of features we could potentially compute is comically large, including such things as "the number of odd lot trades happening in an even numbered second at a price that is within $\frac{1}{3}$ of the spread from the NBB and ... ."

The trick, of course, is deciding what features we should focus on. In defining features of trading behavior to study, we have several goals:

- The features should be likely to be related to price movements.
- The features should be simple and common enough to enable robust modeling in noisy data sets.
- The features should capture the important effects of our algo's trading that are "visible" to the market.
- The features should be reliably related to our scheduling decisions.

The first two goals concern the *tractability* of the data science problem we will be tackling as we try to model the relationship between features and our Δ_i time series. In general, more complicated features and models require more and more data to train reliably, and are less robust in the face of changing conditions. Hence we prefer simple features taking on commonly occurring values, so we can collect lots of examples in training data. Naturally, we also need these features to actually have a meaningful relationship with the price sequences we are representing with Δ_i values.

The last two goals concern the *relevance* of the data science problem we will be tackling. If the features we study don't really capture the relevant effects of our trading, or aren't things we can at least somewhat control through our scheduling decisions, they won't be a good foundation for the decision-making process of our impact-minimizing scheduler. As an extreme example, we could tautologically define Δ_i as a feature, and viola! It perfectly correlates with Δ_i! But clearly this does not provide us with any insight about how to schedule our trading volumes to minimize our influence on Δ_i's. However, if we have a feature like "the volume of trades that occurred at the NBO as a percentage of average daily volume," it is reasonable to expect that this might have a meaningful relationship with price, and that we have some control over how our trading contributes to it. We might use what we can learn about the relationship between this feature and price to help make scheduling decisions, as we might project that scheduling a certain amount of volume to buy will lead us to take a certain portion of that volume at the NBO, and hence may influence Δ_i in accordance with the typical relationship between our feature and Δ_i. Once we have a way of predicting how our scheduling decisions affect features, and a way of predicting how those features affect price movements, we can start to combine these to predict the expected impact of our decisions, and hence meaningfully compare different schedules we might choose. Then we can ultimately choose a schedule that minimizes our expected impact, conditioned on completing the target amount of volume in a specified time frame.



There are some possible pitfalls here that we must remain keenly aware of. One is the classic "correlation does not equal causation." We are basically hypothesizing that our scheduling decisions lead to quoting/trading behavior, which then causally impacts price movements. However, the relationships we find between behavioral features and price movements in historical market data are correlations, and it's possible they are not causal in the way that we intend. It's also worth noting that the two flavors of goals for our feature selection, tractability and relevance, are in tension to some extent. When we think about trying to capture everything about our own actions that might drive an impact in price, it's easy to come up with a very long list of possible features. It might feel intuitive from this perspective to throw in everything that we can, hoping that the union of all the things we've thought of does a good job of covering all the bases. The level of noise in price movements, however, is likely to render this approach untenable. Instead, we'll need to carefully budget and capture as much relevance as we can in a small package of features in order to give us a chance at building robust and meaningful models. All of these challenges (and more!) make feature selection very fraught. As a result, we expect to be continually revisiting and attempting to improve this stage of the research as we iterate on our algorithm's design.

For a candidate set of features, we'll investigate questions like: how strong is the apparent relationship between these features and the time series of Δ_i values? How strong is the evidence for the apparent relationship? How stable does the relationship appear to be over time and market conditions? How reasonable is it to hypothesize that the relationship is at least somewhat causal and will hold up as a model for how our actions may influence prices? How directly do the features capture trading behaviors that are linked to our scheduling decisions?

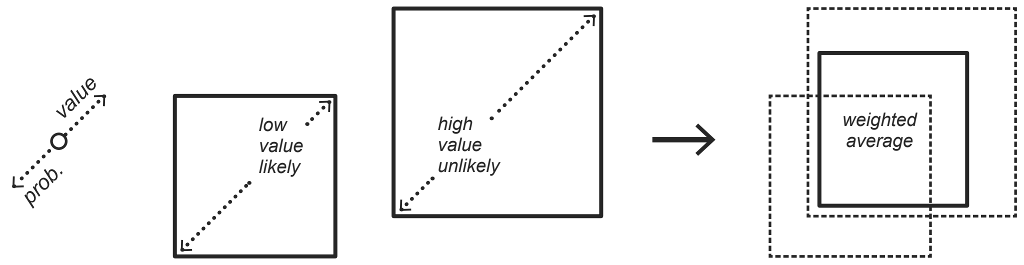
Let's skip ahead a bit and assume we have reasonable answers to those questions for some set of features. Not great answers - we don't want to oversell it - but reasonable ones. How will we go about building a scheduler from this? At this point, let's assume we have what we think are reasonable models for how the features of trading and quoting behavior we've selected affect price movements, and a reasonable model for how our scheduling decisions drive our contributions to those features. What's still missing though, is a model of the wider market's contributions to the features. We could try to predict the market's contribution based on the information we have historically and in real time, but we shouldn't really expect to predict a single value with confidence. For the very basic feature of trading volume, we do have some initial research on predicting this, and it is used in our existing VWAP algo (see https://www.prooftrading.com/docs/vwap.pdf for more details). But for different features, especially more complex ones, developing meaningful prediction models is likely to be a separate new and challenging research task. Also, connecting multiple layers of models together in order to output final scheduling decisions can be precarious, and may lead to amplification of discrepancies between the models and reality.

For now, we prefer to take a more agnostic approach and model the contribution of the rest of the market as a random process. What this will boil down to is averaging over market conditions in our historical market data to determine the expected impact of our actions, and not allowing ourselves to be overly swayed by real time information that is very noisy. We can think of the wider market as a random process that serves up market conditions for each interval, and we'll try to compute the expected costs of our possible scheduling decisions while accounting for this randomness. In other words, we can try to live with a high degree of unpredictability in wider market behavior, rather than immediately trying to tame it with further prediction models that output a single guess for a state of the market.

Whatever kind of random process we choose as our mental model of the wider market is going to be heuristic and deviate significantly from reality. In choosing our model of general market operation, we face similar tensions to what we face in choosing the features for our model of price impact. If we choose a more complex model, it can potentially capture more and more of the nuances of real market conditions and their evolution. If we choose a simpler model, we are more likely to be able to fit its parameters in a robust and meaningful way with a reasonable amount of historical market data. To navigate this tension, we begin with a very simple model that is still capable of modeling the basic market dynamics that we know we want to represent: price impact and price reversion. We discuss this in detail in section 6.

Assembling all of these pieces together, we have: 1) a firm definition of features of trading behavior that we believe can causally influence prices, 2) a model of how our scheduled trading activity contributes to these features, 3) a model of how general market behavior contributes to these features, and 4) a model of how these features contribute to price movements. Using these in combination, we can begin to project the expected costs of our possible scheduling decisions.

We should note here that the meaning of "expected" is the probabilistic one - where we average over the costs of different possibilities weighted by the probabilities we assign to them occurring. This kind of averaging can violate our intuition about what "expected" means in a colloquial sense. For example, a purchased lottery ticket will either turn out to be worth nothing in the likely event that it wins, or worth a lot in the unlikely event that it wins. Averaging these two possibilities yields a modest positive expected value which doesn't correspond to either scenario in isolation. When we talk about mathematical "expectations," we aren't talking about what we "expect" to occur in any particular case. We're talking about a weighted average over the possible cases for an event that is nondeterministic. This weighted average corresponds more closely to reality when we think about many repetitions of the same circumstances, sampled over and over again with fresh randomness. But for any one particular instance, the expectation may be pretty far off from what precisely happens. Ironically, that may even be "expected!"



In designing trading algorithms, we are working in a setting where we will be facing the same kind of circumstances over and over again, so making decisions based on what's best "on average" seems quite reasonable. However, even our computations of averages are likely to be somewhat off due to noise, especially when we are trying to compute averages for market conditions or actions that may be rare. This is a reason to place strong guard rails around the possible scheduling decisions we consider with this methodology. The quality of our projections of expected costs for possible scheduling decisions is likely to degrade quickly as we deviate further from common, relatively low participation rates. Hence this kind of modeling approach should not be relied up to evaluate proposed schedules that include heavy trading flow for which it is hard to find a wealth of proximate examples in historical market data.

This means that we need one more piece in place to translate the outputs of our research process into a scheduler for our algo: we need a constrained set of possible schedules to consider and compare. One constraint is clear: we should only consider schedules that complete the total amount of trading we want to complete by the end of the allotted time. We can also add caps on how much is scheduled in any particular time interval, say as a percentage of the average daily volume (ADV) for that symbol. If we want, we can make such caps a function of what time of day it is, or of the historical volume curve at that time, etc.

Once we have a set of schedules that we want to consider and we feel reasonably good about our ability to project expected costs for the schedules in this set with our research-generated models, it seems we are in good shape! Easy, we might say. Let's just compute the expected cost for each of the reasonable schedules in our set, and then follow the schedule that has the lowest expected cost. But not so fast, there is one remaining challenge.

The challenge is that the set of "reasonable" schedules is still too large for us to exhaustively compute the expected costs for each schedule in it. As an illustrative example, imagine that we divide the regular day into thirty-nine time intervals, each lasting ten minutes, and for each interval we consider four possible amounts that we could schedule. Even if the last interval becomes determined because we have to just schedule whatever we have left at that point, this still represents $4^{38} = 2^{76}$ possible schedules. That's just way too much for even modern computers to handle. A general rule of thumb for gauging what computations are feasible is: 2^{10} is nothing, 2^{20} is something, 2^{50} is going to require specialized hardware, 2^{80} is probably out of reach, and 2^{272} is about how many atoms there in the universe. So we'd like to keep our computational burdens down in the 2^{20} range or below if possible.

Luckily, there are ways to find the schedule with the lowest expected cost without needing to compute the expected costs for every plausible schedule. One such way is called "dynamic programming." The main reason this works for our task is that the lowest cost schedule we are looking for has a lot of convenient properties. Namely, if the lowest cost schedule trades X units of volume in the last Y time intervals, than however it accomplishes that corresponds to the lowest cost way to trade X units in the last Y time intervals. This means we can break the problem into smaller pieces, solve for the lowest cost options for those individual pieces, and then begin to assemble our solutions back into a full solution to the total scheduling problem. This allows us to dramatically reduce the computational resources required to find our solution. We go through this in more detail in Section 7.

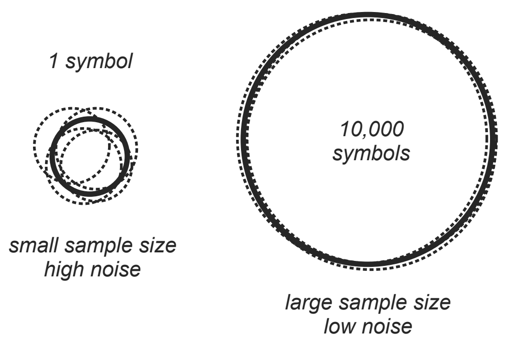
This is especially important because we would like our cost estimates for proposed schedules to be able to depend on real-time information. In particular, when we are making a scheduling decision for the next time interval, we can know information about our recent trading that we couldn't have known at the beginning of the day. We can know, for instance, how our most recent volume translated into the features that we are tracking. For a feature like "the amount of volume trading at the NBB," for example, our contribution to that feature varies not only as a function of how much volume we schedule, but also as a function of how quickly we are able to pick up passive volume and how frequently we end up needing to the cross the spread. These are not things we will know with certainty until the interval is over. If we can reasonably compute new cost estimates for proposed schedules of the remaining volume in real-time, we can take advantage of information like this once its known to us, and potentially make choices for the upcoming interval that are better informed. This will only work if the computation of the future schedule with the lowest expected cost is fast enough to be continually performed on the fly during the trading day.

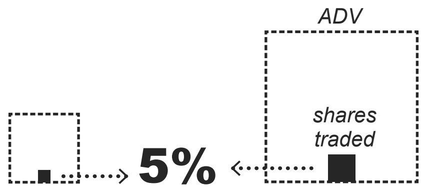
In the subsequent sections of this paper, we will describe the research underpinning our choices for each piece of the infrastructure described above. This represents the initial form of the scheduler we have designed to try to minimize price impact, subject to completing a specified target amount of volume. We only invoke this scheduler for volumes that are capped to a certain limit of the symbol's ADV, in order to avoid relying on our models in cases beyond our perception of their reliability. We expect this is merely the preliminary form of an algorithm that we will quickly iterate on, and merely the beginning of a long research agenda targeted and understanding price impact and how to minimize it for our trading.

3 Data Normalization

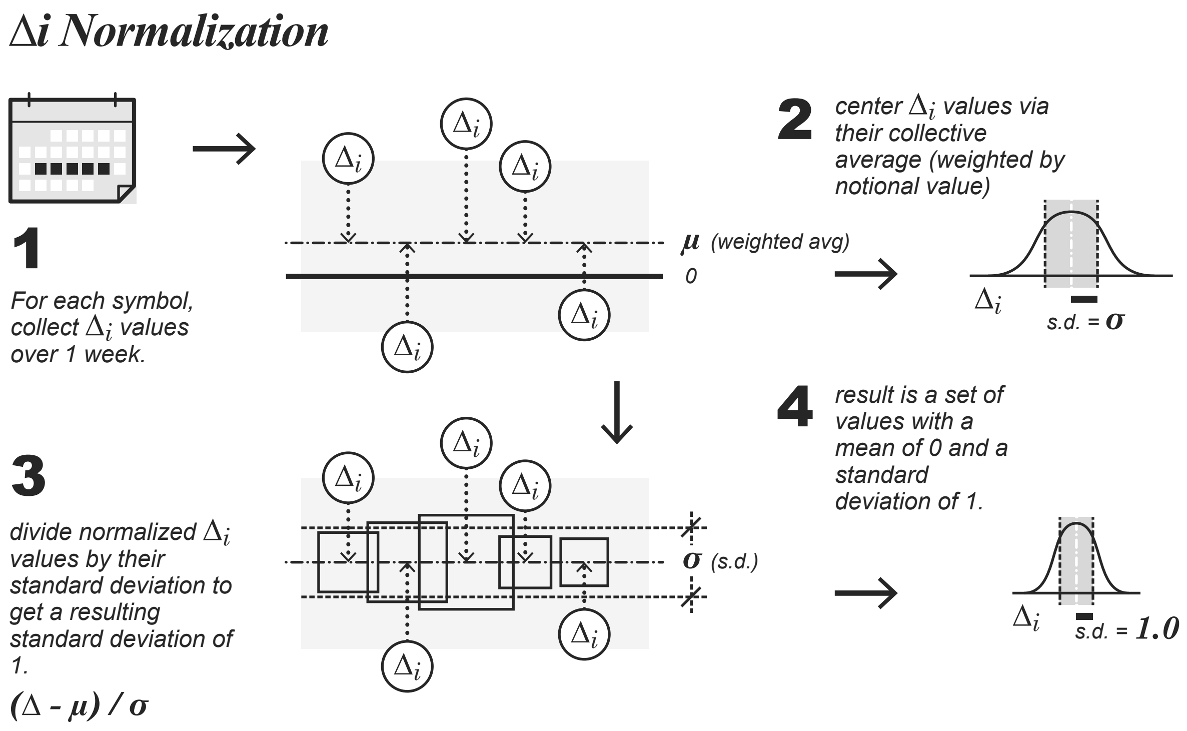
As of the time of this writing (fall 2021), there are over 10,000 symbols trading on the US equities markets. Hence the sample size of historical market data available for each individual symbol is several orders of magnitude smaller than the sample size available if we accumulate over all (or at least many) symbols. For modeling price impact, we suspect that the level of noise in price data will be nearly overwhelming, and we'll need all of the samples we can possibly get. Hence, we will try to accumulate data over symbols for developing and training our models.

Doing this requires several strategic decisions about how to normalize data so that it can be meaningfully combined across symbols. Different symbols may trade at very different price levels, in very different amounts. To make things a bit more apples-to-apples, we'll always look at size quantities relative to the ADV in a given symbol. In other words, instead of considering raw counts of shares, we'll divide those counts by the ADV and consider percentage of the ADV as our primary unit of size. [Technical note: we calculate ADVs as averages over the set of trading days included in the last 20 calendar days. We've considered slight variations on this, like using 20 trading days or 30 calendar days, etc., and we've not found any compelling reasons to prefer one over the others.]





For prices, the Δ_i values we have defined above already provide some normalization, as they consider relative price changes over time intervals rather than absolute dollar amounts. However, different symbols will exhibit different levels of variance in their typical Δ_i values. General market trends will also contribute to Δ_i values, making them unlikely to be mean 0 over time periods where the market was generally up or generally down. Since our purpose here is to understand price impact at a more localized level, we choose to take all of the Δ_i values for a given symbol over a given time period in our data set and center them at 0 by subtracting their collective average from each of the individual values. We also divide by their collective standard deviation in order to force the standard deviation of the normalized values to be 1. In other words, we are taking a set of Δ_i values that has an arbitrary mean and standard deviation and forcibly scaling and adjusting it to have a mean of zero and standard deviation of one. We do this for the Δ_i's of each symbol individually, over time periods of 1 week at a time. When we compute the mean and standard deviation of a collection of Δ_i values, we weight the individual values according to the notional value traded in each interval. (It would also be reasonable to weight all of the values equally for a given symbol in a given week, but we generally weight things by notional value unless there is a compelling reason to weight them otherwise.)


Δi Normalization
1
For each symbol, collect Δ_i values over 1 week.
Δ_i
μ (weighted avg)
0
2
center Δ_i values via their collective average (weighted by notional value)
Δ_i
s.d. = σ
3
divide normalized Δ_i values by their standard deviation to get a resulting standard deviation of 1.
$(\Delta - \mu) / \sigma$
σ (s.d.)
4
result is a set of values with a mean of 0 and a standard deviation of 1.
Δ_i
s.d. = 1.0

Performing this normalization on Δ_i values has several consequences. One is that we have forfeited any ability to model price impact at time scales of a week or longer. In exchange, we have greatly reduced the confounding influence of wider market trends that obscure the targeted effects we are attempting to model. Since here we are focused on designing a schedule for intra-day behavior, we think this is a worthwhile trade-off. But doing this kind of forcible transformation to make things mean zero and standard deviation one is not the only way we could try to reduce confounding factors. Another approach would be to use our *distilling* techniques, developed in our prior white papers on the topic of distilled impact (available at: https://www.prooftrading.com/#section-research). We use these distillation techniques currently to attempt to reduce the influence of wider market forces in our TCA calculations. However, the amount of noise that can be removed through these methods remains smaller than we would like it to be. We do not think that our distillation methods are currently strong enough to normalize data across symbols for the tough task here of training models relating noisy price data to behavioral trading features. We hope this will change in the future as we improve our distillation techniques, and our impact modeling tools.

Once we have normalized data for each symbol and each time period in our data set, it remains to decide how to aggregate data across symbols. Here again, we choose to weight each data point proportionally to its notional value. This ensures that data for symbols whose trades cumulatively represent larger notional value will contribute more strongly to our modeling than symbols whose trades cumulatively represent smaller notional value. In essence, we are behaving here as if we expect that our own trading will ultimately be distributed across time and symbols similarly to the distribution of notional value across the market.

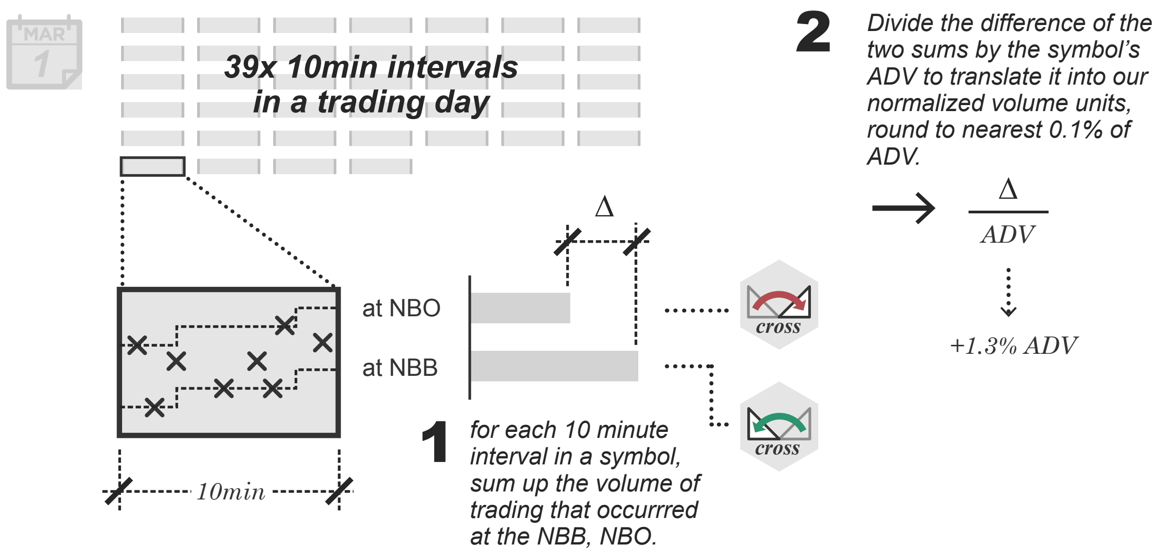
4 Feature Selection and Modeling Price Movements

Deciding which features to compute, test, and ultimately keep as part of a model is typically the hardest and most important part of any data science research. This project was no exception. In fact, our feature selection process spanned well over a year, involved a lot of false starts, and was ultimately re-invigorated by the launch of our VWAP algo. Watching our first algo's tactics in action spurred a few clearer ideas for how we might best capture our low-level trading actions in simple features that we can also compute on historical market data.

Our algo's tactics involve two main behaviors: posting and taking. When we cross the spread and take as a buyer, for example, we trade at the current NBO price. When we post as a buyer, we join the current NBB price. Both of these actions, though in different ways, potentially signal to the wider market that there is additional buying interest and may ultimately drive prices up. To try to capture the effect of our taking behavior, we define a feature on trade data that accounts for spread-crossing trades. To try to capture the effect our posting behavior, we define a feature on quote data that accounts for events where size increases at the NBB/NBO.

More specifically, we start by labeling each reported trade as occurring at the prevailing NBB, at the prevailing NBO, or neither. For each 10-minute time interval and each symbol, we sum up the volume of trading that occurred at the NBB, as well as the amount of trading that occurred at the NBO. We compute the difference of these two sums, and then divide by the ADV to translate it into our normalized volume units. This gives us one number whose sign represents which side of the NBBO has experienced more trading, and who magnitude reflects the size of the difference, relative to the ADV.

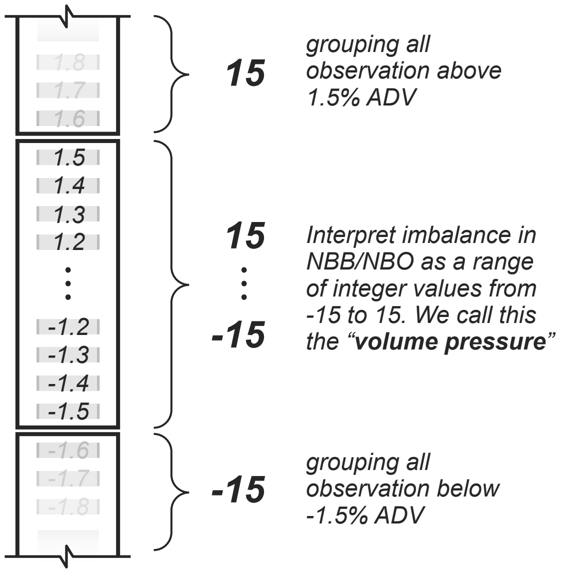
Volume at NBB/NBO Feature


MAR
1
39x 10min intervals in a trading day
2
Divide the difference of the two sums by the symbol's ADV to translate it into our normalized volume units, round to nearest 0.1% of ADV.
Δ
ADV
+1.3% ADV
Δ
at NBO
at NBB
cross
cross
10min
1
for each 10 minute interval in a symbol, sum up the volume of trading that occurrred at the NBB, NBO.

Our approach to evaluating candidate features involves grouping together time period/symbol combinations that have similar values for the features in training data and then seeing how the (notional value weighted) average Δ value of these behaves as a predictor for the Δ value of fresh testing data with those feature values. This grouping process is a very basic kind of model that we can evaluate directly without having to do slightly fancier things like fitting a linear function or a decision tree to the relationship between the features and Δ values. This grouping will only work well if we scale and round values in such a way that the groups contain sufficient sample sizes, but also don't conflate too many disparate situations and hence obscure the features' influence. This can be a tricky balance to achieve, but we can try a few different roundings/groupings and see which ones provide better quality predictions on the testing data. As a starting point for our volume at the NBB/NBO feature, we'll round to the nearest 0.1% of ADV. In other words, the nearest multiple of $0.001 * ADV$.

We also suspect that the difference between a value of, say 2.0% ADV and a value of 2.1% ADV for this feature may be less meaningful than the difference between a value of 0.1% ADV and a value of 0.2% ADV. As the values get higher in magnitude, we are likely to see sample size at each 0.1% increment drop off sharply, and the meaningfulness of these distinctions degrade. To reduce the clutter and overhead in our computations over large swaths of historical data, we capped our feature value at 1.5% ADV, grouping together all observations at +1.5% ADV and above, and grouping together all observations -1.5% ADV or below. With this cap in place as well as the rounding to the nearest multiple of 0.1% ADV, we can ultimately view this feature as an integer ranging from -15 to 15. We call this feature "volume pressure," and we think of it as trying to capture the "pressure" exerted on price by the (im)balance of trading volume happening at the NBB/NBO.

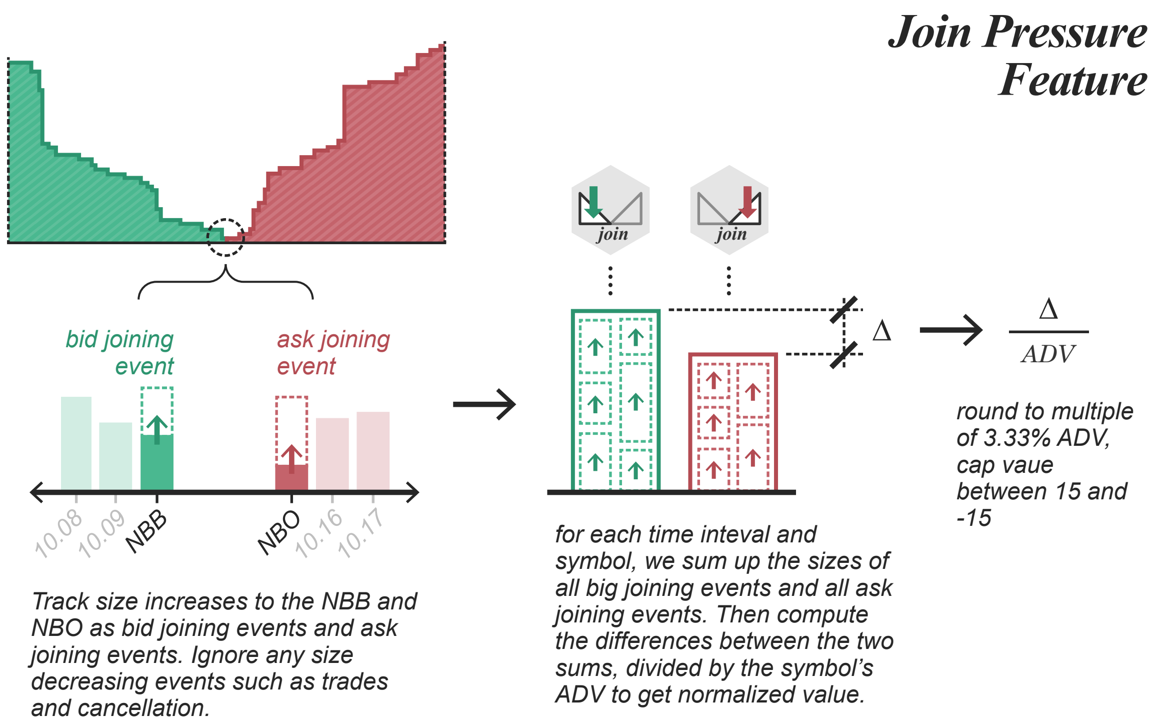
Capping Volume Pressure


1.8
1.7
1.6
15
grouping all observation above 1.5% ADV
1.5
1.4
1.3
1.2
-1.2
-1.3
-1.4
-1.5
15
-15
Interpret imbalance in NBB/NBO as a range of integer values from -15 to 15. We call this the "volume pressure"
-1.6
-1.7
-1.8
-15
grouping all observation below -1.5% ADV

We also define a feature that we compute from the top-of-book quotes within each 10-minute time interval for each symbol. First, we use the top-of-book quotes to construct the NBBO at each moment in time. In this process, we construct the sizes available at the NBBO as well the prices, and we keep track of all size changes to the NBB or NBO, even when they do not represent price changes. Each time the size at the NBB increases while the price stays the same, we label this as a "bid joining event," and we define the size of the event to be the amount of increase in the size available at the NBB. Similarly, each time the size at the NBO increases while the price stays the same, we label this as an "ask joining event," and we define the size of the event to be the amount of increase in the size available at the NBO.

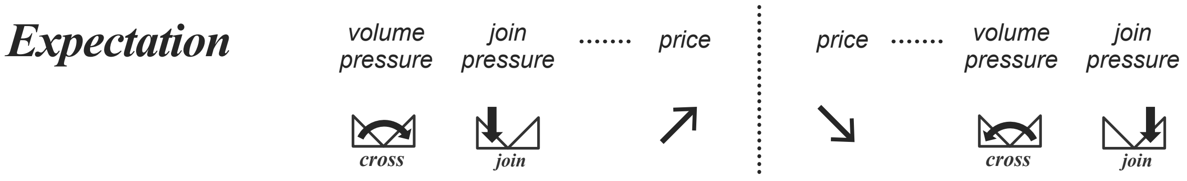
For each time interval and symbol, we sum up the sizes of all the bid joining events and all of the ask joining events. We compute the difference of these two sums, and divide it by the ADV. This gives us a number whose sign indicates which is more popular to join, the NBB or the NBO, and whose magnitude reflects the size of the difference. A couple things to note here: first, we do not consider the establishment of a new price level to be a "joining" event. Second, we do not track events where size decreases. For example, if the NBB stays at the same price but the size available changes from 2 lots to 6 lots, then to 4 lots, then to 7 lots, there are two NBB joining events in that sequence: the increase from 2 lots to 6 lots, and the increase from 4 lots to 7 lots. These will contribute a total of $+4+3 = +7$ lots to the sum of NBB joining events. We do not track the decreases or whether they represent trades or cancellations, etc.

Like our volume pressure feature, the difference between the two resulting sums (normalized by ADV) is rounded and capped. This time, we round to the nearest third of a percent of ADV, i.e. $0.00333\ldots$ ADV. (We rounded this a bit more coarsely because the numbers tend to be a little bigger than the traded volume numbers were, so we seem to get similar differentiation with a slightly coarser rounding.) After rounding and translating to integer multiples of 0.0033 . . . ADV, we again applied a cap of 15 on the magnitude, grouping together all observations beyond this cap. We call this feature "join pressure," and we think of it as trying to capture the "pressure" exerted on price by the (im)balance of demand indicated by parties joining onto the NBB vs. the NBO.

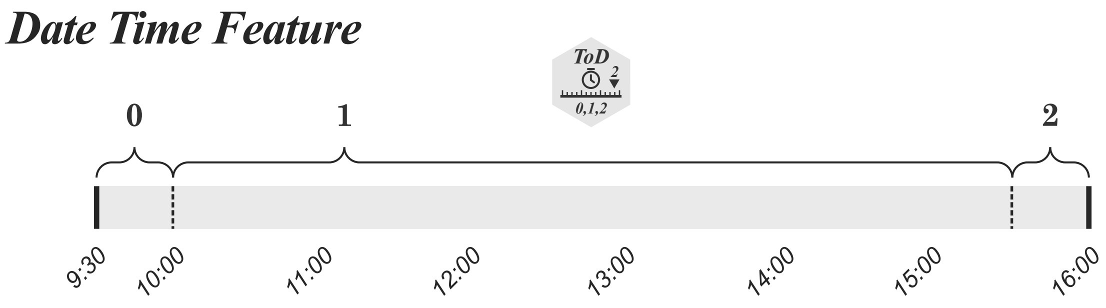


Join Pressure Feature
bid joining event
ask joining event
10.08
10.09
NBB
NBO
10.16
10.17
Track size increases to the NBB and NBO as bid joining events and ask joining events. Ignore any size decreasing events such as trades and cancellation.
join
join
Δ
Δ / ADV
for each time inteval and symbol, we sum up the sizes of all big joining events and all ask joining events. Then compute the differences between the two sums, divided by the symbol's ADV to get normalized value.
round to multiple of 3.33% ADV, cap vaue between 15 and -15

There are a lot of nuances to the definitions of our volume pressure and join pressure features, and there are many similar but slightly different ways we could have defined them. We suspect that most of those variations would yield similar results. We choose these versions because they keep closely in spirit to our own trading behavior. We choose these roundings and caps because they seem to lead to a level of aggregation that is manageable for computing over several months of historical market data, and as we will see shortly, a level of precision that is likely more granular than we ultimately need. Indeed, we will further round more coarsely later in order to reduce over-fitting.



We expect that more volume at the NBO vs. the NBB and more joining at the NBB vs. the NBO will generally push prices up, and the opposite signs will generally push prices down. We also suspect that price impacts will vary a bit throughout the day: in particular, we would like to account for different effects in the morning just after the open and nearing the end of the day going into the close. For this, we add a very simple feature that takes three values: 0,1,2. We mark as "0" any time periods that are before 10 am, as "1" any time periods between 10 am and 3:30 pm, and as "2" any time periods from 3:30 pm until 4 pm.



We also suspect that there will be reversion effects. If one of our pressure gauges is dialed up for one time interval and then dialed back down in the next, we might guess that the price will revert a bit, rather than responding merely to the new pressure value in isolation. To enable us to see and model this kind of effect, we add features for the values of the volume pressure and the join pressure in the previous time interval.

Putting this all together, for each ten minute time interval and each symbol, we compute eight things:

1. the volume pressure for the trading in that interval, as an integer from -15 to $+15$.

2. the join pressure for the quoting in that interval, as an integer from -15 to $+15$.

3. the volume pressure for the trading in the previous interval, as an integer from -15 to $+15$.

4. the join pressure for the quoting in the previous interval, as an integer from -15 to 15.

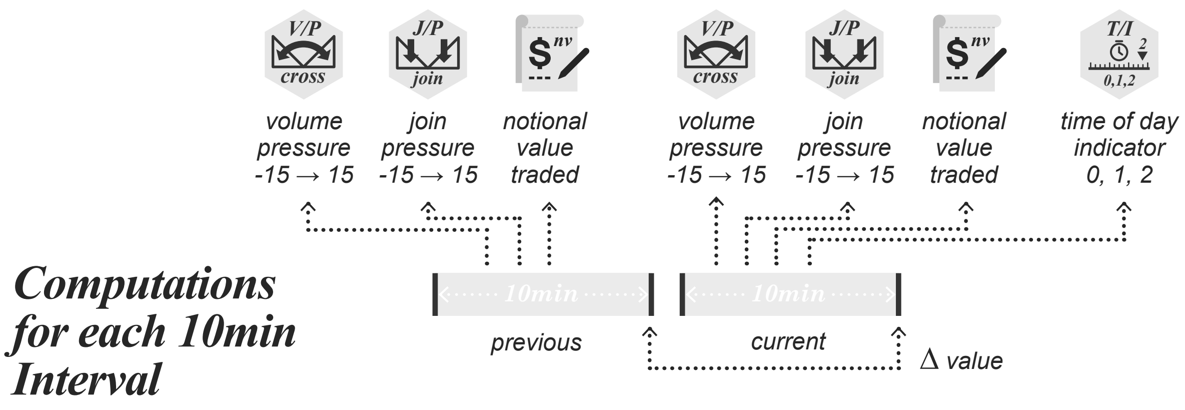
5. the time of day indicator, as an integer from 0 to 2.

6. the Δ value, i.e. the natural logarithm of the ratio of the last trade price in the interval over the last trade price in the previous interval.

7. the notional value traded in this symbol and interval.

8. the notional value traded in this symbol in the previous interval.

The last two of these items, the notional values, are computed because we will be using them later for weighting purposes.



Since several of these items mention "the previous interval," it is not clear what to do for the first ten-minute interval of the day, from 9:30 am to 9:40 am. One reasonable option would be simply exclude these intervals, since they don't have well-defined values for all of the features we are attempting to study. Another reasonable option would be to define the previous pressure gauges as zero, and use the first trade price in the interval as the denominator for Δ instead of the last price of the previous interval. Currently we just exclude them.

Once we have collected these values across all of the 10-minute time intervals for a given symbol in a given week, we compute the average and standard deviation of the Δ values (weighting by notional value). Employing common statistical notation, we'll use μ to represent the weighted average (aka expected value), and σ to represent that standard deviation. We'll then normalize our Δ values by performing the following transformation:

$$\tilde{\Delta} := (\Delta - \mu)/\sigma.$$

Going forward from this point, we will work only with the adjusted $\tilde{\Delta}$ values and can discard the original Δ values. We note that μ is a term representing overall price movement, and will not show up as part of our later impact calculations. But σ is a term that roughly represents price volatility, and will reappear later as we translate our general model of the relationship between trading behavior and prices into expected costs of contemplated behavior in a particular symbol. In this way, we are enabling our model to account for the fact that the magnitude of price impact can be greater when volatility is greater.

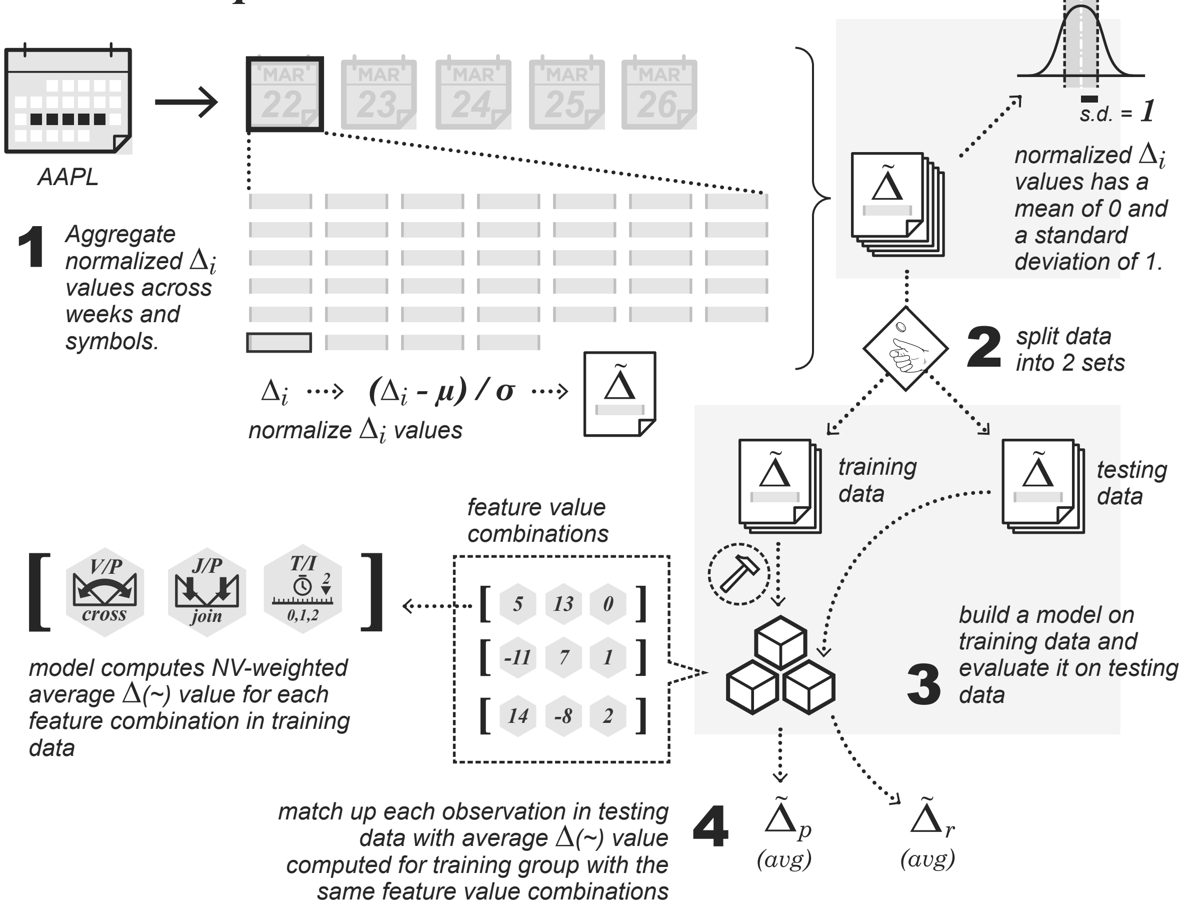
Our next step is to aggregate all of this collected and normalized data across weeks and symbols into two data sets: one for training our feature models, and one for testing the quality of the trained models. It is simplest to make these data sets disjoint, so that we can interpret the results without having to be concerned about nuances that arise when the data sets are overlapping. [Aside: training and testing on the same data is a bit like leaking the exam questions to students before the test. The results become much less representative of true knowledge.] We could do this by collecting the training and test data over disjoint weeks or months. Another simple way is to flip a random coin for each data point to decide if it should be assigned to the training or the testing data set. This may result in the training and testing data sets being more similar to each other, as they are not covering disparate time periods that may have different market dynamics. Since we expect feature selection and modeling to be a hard task in this noisy environment, we choose for now to do the coin flip method, and flip the coin independently for each set of 7 collected values. This is a little weird in our case, as our features about the prior interval mean there is some dependence between adjacent observations. But we do not expect this to cause any significant issues. We could address this by fixing the outcome of the coin flip at the level of symbol/day, but there would still be some implicit dependence across days in the same symbol due to our adjustment of the Δ values. This is a rabbit hole we choose not to go down.

Once we have our training and testing data sets, we are ready to build a basic model on our training data and evaluate it on our testing data. We'll start with a model that just computes a notional value weighted average $\tilde{\Delta}$ value for each combination of feature values in our training data. To evaluate the model on our testing data, we'll match up each observation in our testing data with the average $\tilde{\Delta}$ value computed for the training group with the same combination of feature values. We'll refer to this average $\tilde{\Delta}$ value as $\tilde{\Delta}_p$ (where the p denotes that is our prediction, based on training data). The real $\tilde{\Delta}$ value for the testing data point we'll denote as $\tilde{\Delta}_r$. The squared error of our prediction for this data point is then computed as:

$$(\tilde{\Delta}_p - \tilde{\Delta}_r)^2.$$

We'll average these squared error values over all data points in our testing set, weighting them by notional value. We'll refer to this basic style of model as a "table lookup" model.

Table Lookup Model



AAPL
MAR 22
MAR 23
MAR 24
MAR 25
MAR 26
1 Aggregate normalized Δ_i values across weeks and symbols.
Δ_i ···> $(\Delta_i - \mu) / \sigma$ ···> $\tilde{\Delta}$
normalize Δ_i values
s.d. = 1
normalized Δ_i values has a mean of 0 and a standard deviation of 1.
2 split data into 2 sets
training data
testing data
3 build a model on training data and evaluate it on testing data
feature value combinations
[5 13 0]
[-11 7 1]
[14 -8 2]
V/P cross
J/P join
T/I 2 0,1,2
model computes NV-weighted average Δ(~) value for each feature combination in training data
4 match up each observation in testing data with average Δ(~) value computed for training group with the same feature value combinations
$\tilde{\Delta}_p$ (avg)
$\tilde{\Delta}_r$ (avg)

Once we have an NV-weighted average squared error, how do we know if it is "good" or "bad"? Clearly, if we compare two models to each other, we would prefer the one with the lower of the two average squared errors. We should not neglect though, the necessity of including a very simple baseline. Sometimes all of our more complicated models are "bad," and we shouldn't blindly take the best performing one as meaningfully good. For this case, our baseline will be a "model" that merely predicts 0 as the $\tilde{Delta}_p$ value, regardless of any of the feature values.

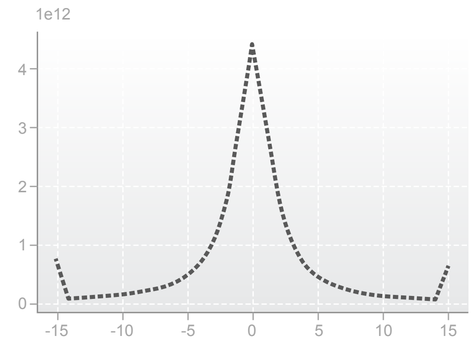
We collected data for the top 1000 symbols in terms of notional value over the three month period from April 1, 2021 to June 30, 2021. Computing our features over this set of symbols and days takes a considerable but reasonable amount of computation power. Since we are weighting everything by notional value anyway, we don't expect that it would be worthwhile to compute the features over all of the remaining symbols, given that it would take a large amount of computational resources and the top 1000 symbols typically represent the majority of the notional value traded.

We split this data into roughly equally sized sets as training and test data by assigning each data point to one or the other uniformly at random. As a first test, we left our pressure features all at the granularity of a scale from -15 to 15 and computed the NV-weighted squared error for our baseline as well as our basic (aka "table lookup") model. At this granularity, some combinations of features values occurred only in the test data or only in the train data, making the lookup approach quite precarious. Even generously limiting ourselves to the cases where the test data features *did* appear in the training data, the model's squared error was ... drum roll please ... 1.08 times the baseline. So this does *worse* than nothing basically. But hey, at least we worked really hard for it!

This isn't too surprising, and doesn't mean we have to go back to the drawing board (though we did that too a few times over the course of this research). An integer scale from -15 to 15 means that each pressure gauge can take on any one of 31 values, and we actually have four pressure gauges: two for the volume and join pressure in the current interval, and two for the previous interval. These means that the pressure gauge values alone can potentially drive the lookup table size to be as big as: $31^4 = 923521$. Having a table with this many possible values can unsurprisingly lead to the sample size behind each value often being insufficiently small. Once the data is collected though, it's easy to regroup the data more coarsely along the same features to make the table smaller and the samples behind each value larger. (That's why it's typically wise to start with something too granular. If you start with collecting data that is already rounded to be too coarse, you have to recollect all of the data from scratch to get greater precision.)

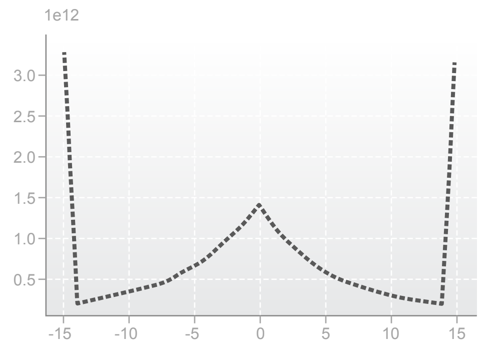
As a next step, we can try grouping our ranges from -15 to 15 into coarser categories, using multiples of 5 as boundaries. In other words, we'll group together the values 11, 12, 13, 14, and 15, and group together the values 6,7,8,9,10, etc. We'll let 0 be its own group. If we do this, essentially all combinations of values occur in both the test and training data, and we get a model whose squared error is about 0.86 times the baseline. This is looking much more promising - at least we're beating the baseline!

We may pause for a moment here and wonder though - should we be grouping higher values together in the same way as we are grouping lower values? This uniform approach to grouping makes a lot of sense when values are somewhat uniformly distributed, but that may not be the case for our data. Let's take a look and see! At the finer scale from -15 to 15, let's plot the total notional value that occurs over our data set for each of our two pressure gauges. Here's the plot for volume pressure:



This shows a concentration in smaller values, and suggests that we may want to aggregate less coarsely when we are close to 0 and more coarsely as we get out to higher absolute values. Note that the visually strange behavior of curling up at -15 and 15 is expected, since we capped the volume pressure values. The notional value on these end points represents the accumulation of all notional value at *and beyond* the end point, so we are seeing the total size of the tails here.

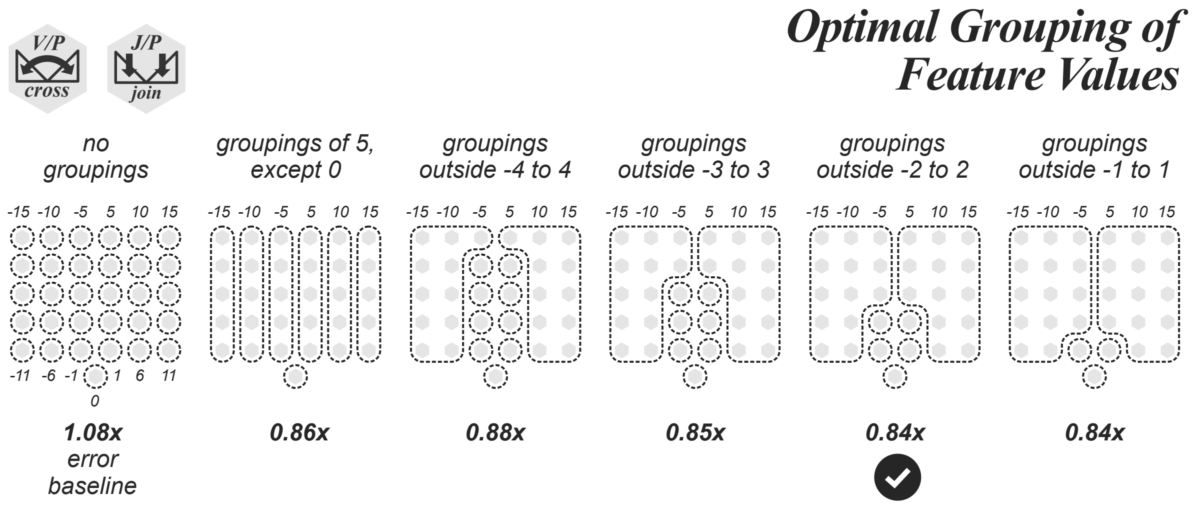
Here's the same plot for join pressure:



This also shows some concentration toward lower values, but the accumulated tails are much larger in proportion here than they were in the volume pressure distribution. This is a notable phenomenon that could be interesting to further investigate.

For now though, we'll just try to define a reasonable grouping scheme that zooms in on the concentrated region near 0. A very simple way to do this is to make the caps much lower than 15. For example, we could group together all of the values of 5 and above (and group together all of the values of -5 and below), while retaining the distinctions between all values in the range from -4 to 4. As it happens, this particular choice does better than our baseline, but a little worse than our original uniform grouping. It achieves a squared error that is about 0.88 times the baseline.

If we get a little more aggressive and move our caps all the way to -3 and 3 respectively, however, we do slightly beat our uniform grouping and achieve a squared error that is about 0.85 times the baseline. If we move them even further to -2 and 2, we get a squared error that is about 0.84 times the baseline. If we go all the way to capping at 1 (i.e. distinguishing only between 0, positive values, and negatives values), we get a squared error that is 0.85 times the baseline again. This indicates that nearly all of the predictive power of these features is contained in their signs, rather than their magnitudes. We'll fix the caps at -2 and 2 for now, as this performed the best of the options we tried.


Optimal Grouping of Feature Values
V/P cross
J/P join
no groupings
groupings of 5, except 0
groupings outside -4 to 4
groupings outside -3 to 3
groupings outside -2 to 2
groupings outside -1 to 1
-15 -10 -5 5 10 15
-11 -6 -1 1 6 11
0
1.08x error baseline
0.86x
0.88x
0.85x
0.84x
0.84x

We should also check if any obvious subsets of our features contains as much predictive power as the full set. For instance, should we really be including both pressure gauges, or is it as good to have only one? It would be worthwhile to explore single pressure gauges that combine our two kinds of pressure, but we leave that as a topic for future research. For now, we confirm that including only our volume pressure gauge and omitting our join pressure gauge would be inferior (we get a model squared error that is 0.89 times the baseline), and similarly that including only our join pressure gauge and omitting our volume pressure gauge would be inferior (we get a model squared error that is 0.91 times the baseline). Omission of our time-of-day feature or of the pressure gauge values in the previous interval causes only a very slight detriment to performance, suggesting that these features are even weaker in predictive power than the two pressure gauges for the current interval (whose predictive power we know is mostly driven by their signs).

It is probably not a great idea for us to continue testing too many possible groupings or subsets of features on this same training and testing data set, as the noise is very high and the relative performance margins are very tight. We are likely to start falling prey to coincidences and over-fitting. So what we'll do now is fix our feature set to signed volume and join pressure gauges capped at an absolute value of 2 for the current and previous intervals and a 3-valued time-of-day feature. We'll perform a robustness test on this feature set by training and testing it on fresh data from a different time period, in order to gain some further confidence that this approach at least performs reliably better than our (very dumb) baseline.

This time, we'll take data from August 1, 2021 through September 1, 2021, and again divide it into disjoint training and testing sets. Using this same feature set yielded a model squared error that was 0.85 times the baseline for this time period, which is eerily close to the performance on our original data set above. [Aside: kind of creepy, but what can you do?]

5 Mapping our Scheduling Decisions to Features

We will also need to build a rudimentary understanding of how our scheduling decisions influence the features we've selected. Directionality at least is clear: if we schedule more shares to buy in an upcoming time interval, we should expect to create some NBB joining events and some taking events where we cross the spread to trade at the NBO. We expect our effect on both pressure gauge features to be in the directions that exert upward pressure on prices. Conversely, when we schedule more shares to sell in an upcoming time interval, we should expect to exert downward price pressure. But exactly how much pressure do we expect to inject on each gauge?

The relationship between our scheduling decisions and the pressure gauges is intermediated by other market participants. If the NBBO moves around considerably while we are posting, we will be periodically repegging and thus increasing our effect on join pressure. If we are not getting enough passive fills, we will be crossing the spread and increasing our effect on volume pressure. Hence, the translation of our scheduling decisions into our contributions to the pressure gauges is driven by the combination of our lower level tactical decisions and the actions of potential counter-parties.

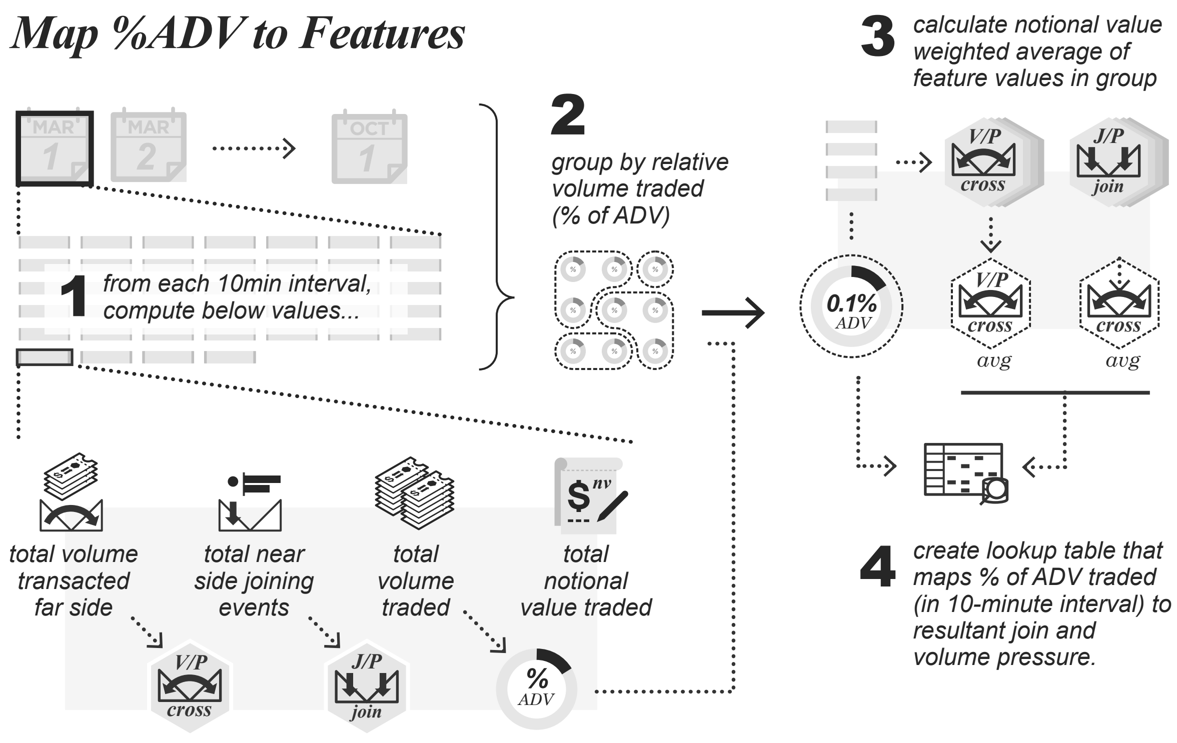
We can learn about average outcomes for this translation process by examining our own recent trading data. For every time interval/symbol/side we traded, we can compute the total volume we traded, the total volume we traded at the far side, and the total size of all our posted orders that joined the near side. Dividing these values by the ADV and adjusting the sign appropriately based on whether we buying or selling, we obtain our contributions to the volume pressure and the join pressure features. In this way, we can collect data points from our own recent trading data to study the relationship between our scheduled quantities and our resulting effects on the pressure gauges. A few technical notes are in order. First, before we launch the trading algo whose scheduler design we are documenting here, all of our live trading data is from the trading we have done using our VWAP algo. We still expect this data to be indicative of what we expect for our new algo on a tactical level, as it is the higher level scheduler that will be different, not the lower level tactics. Second, we may treat traded volumes and scheduled volumes as equivalent for our purposes here, as it is always the intention for the algo to trade its full scheduled quantity within each specified time period. [Aside: if this doesn't happen, it is because of external constraints, like a client's specified limit or a halt in the symbol, etc.]

We have only been live trading since March of 2021, so our trading data so far is fairly limited in quantity. It is unlikely to support overly complex analyses (or really, reasonably complex analyses), so we'll start with something pretty simple. We'll divide each trading day for each symbol and side into disjoint 10-minute intervals, and in each interval we'll compute:

1. the sum of the notional value we traded
2. the sum of the volume we traded
3. the sum of volume we transacted at the far side
4. the sum of the size of our near-side joining events

We also compute the ADV for each symbol, so that we can divide and scale the values of 3. and 4. above to be in the same units as our final definitions of the volume pressure and join pressure features. We also divide the value of 2. by the ADV so that our volumes are in units relative to the ADV rather than in absolute numbers of shares. We round these relative volumes to the nearest multiple of $0.001 * ADV$.

We next group these data points by their rounded relative volumes. What we have now is a group of samples of times when we traded roughly 0.1% of the ADV within a 10-minute time period, a group of samples when we traded roughly 0.2% of the ADV within a 10-minute time period, and so on. Within each group, we can take a notional value weighted average of the volume pressure and the join pressure features. This yields a look-up table that we can use to answer questions like: "when we schedule $X\%$ of the ADV for a 10-minute interval, what amount of join pressure and volume pressure should we expect to create?"


Map %ADV to Features
MAR 1
MAR 2
OCT 1
1 from each 10min interval, compute below values...
2 group by relative volume traded (% of ADV)
3 calculate notional value weighted average of feature values in group
V/P cross
J/P join
0.1% ADV
V/P cross avg
cross avg
4 create lookup table that maps % of ADV traded (in 10-minute interval) to resultant join and volume pressure.
total volume transacted far side
total near side joining events
total volume traded
total notional value traded
$ nv
% ADV

In the next section, we'll get away from modeling only our activity, and try to zoom out to a basic model that considers the extraneous market behavior as well.

6 A View of the Market as a Random Process

In probability theory terms, a random process is one that transitions through a sequence of states, where each state is chosen probabilistically from a distribution that may depend on the prior states. We can think of the current values of our two pressure gauges and the value of our time-of-day feature as together constituting a market state, and the market as a probabilistic process that transitions from one state to the next at regular time increments.

For now, we'll consider a very simplistic kind of transition. The time-of-day feature will transition in a deterministic way as we move through the time intervals of the trading day, and the pressure gauges will sampled freshly and independently at random each time from the conditional distribution over market states with the appropriate fixed value for the time-of-day feature. [Aside: this extremely basic random process as a heuristic model of the market is a prime target for improvement as we continue our research and iteratively build up a more nuanced understanding.]

We can imagine that historical market data gives us a source of information about these distributions *without* our trading in it. More precisely, we can see how notional value typically distributes across the possible states. We can view this as a probability space defined by the random dollar: if we choose a dollar uniformly at random from all the dollars traded, what is the probability that the market was in a particular state i while that dollar was trading? We can answer this question generally, or conditioned on a particular fixed value for the time-of-day feature. These are the kinds of things we can compute by calculating the state and the notional value corresponding to each symbol and time interval in our historical market data set.

Even if the market really were a random process (and it's not!), our computation of these "probabilities" wouldn't correspond exactly to the true probabilities. So we're being a little lazy here when we say we're computing probabilities. What we're actually doing is empirically estimating probabilities *as if* our historical market data was generated according to an underlying random process. Nonetheless, we soldier on.

Next we have to think about how we imagine our trading activity blending into this process. One way we can imagine it is: each interval, the market process tentatively serves up a new random state. Then the effect of our activity is added to the pressure gauges, and these summed features become the actual next state.

Let's now imagine that the trading day evolves as a sequence of steps of our interaction with this random process. In each interval i, we'll let S_i denote the state of the process, which is a vector with three entries containing the values of the two pressure gauges we have defined as well as the time-of-day feature. In other words, $S_i := (V_i, J_i, T_i)$, where V_i denotes the volume pressure gauge value in interval i, J_i denotes the joining pressure gauge, and T_i denotes the time-of-day feature value. We'll let M_i denote the tentative state that is sampled by the process for state i before our activity is included, and we'll let F_i denote a vector containing our contributions to the pressure gauges in interval i. We then have: $S_i = M_i + F_i$. We note that F_i is a probabilistic function of the amount of volume we choose to schedule in interval i. We'll denote that amount of volume as v_i, so we can say that F_i is sampled from a distribution that depends on v_i. We further hypothesize that a price evolution term, $\tilde{Delta}_i$, is sampled for each interval i from a distribution that depends on S_i as well as V_{i-1} and J_{i-1}. [Note: this inclusion of the prior pressure gauges enables us to model price reversion effects from one time interval to the next.]

If we suppose this process is happening for a particular symbol, we can fix a typical standard deviation value σ for this symbol to translate each $\tilde{\Delta}_i$ into a multiplier of $e^{\sigma \tilde{\Delta}_i}$ that represents the ratio between the price at the end of the interval over the price at the end of the previous interval. In other words, we can model the price P_i at the end of i intervals as:

$$P_i = P_0 \prod_{j \leq i} e^{\sigma \tilde{\Delta}_i},$$

where P_0 represents the opening price.

If we are, say, buying volume in this symbol over the course of the trading day, the prices for our trades will vary considerably from these checkpoint P_i values. As a simple approximation though, we will suppose that all of our volume that trades in interval i will occur at the price P_i. This allows us to express the total cost of our purchased shares as:

$$\sum_i v_i * P_i = P_0 * \sum_i v_i \prod_{j \leq i} e^{\sigma \tilde{\Delta}_i} = P_0 \sum_i v_i e^{\sigma \sum_{j \leq i} \tilde{\Delta}_i}.$$

A few things to note here. First, P_0 is just a constant term that we can essentially ignore, so we'll set it to 1 and stop writing it in our expressions going forward. Second, when we say "total cost" here, we are not including the myriad fees that accompany the execution of our transactions, as well as the slippage and variance we would expect to see when comparing our real prices to the P_i values. Third, the v_i's are values we control, but the $\tilde{\Delta}_i$'s are random variables we merely influence through our effect on F_i's which in turn affect S_i's as described above.

Given a fixed set of values v_i, we can define the expected cost function:

$$C(v_1, \ldots, v_N) := \mathbb{E}\left[\sum_i v_i e^{\sigma \sum_{j \leq i} \tilde{\Delta}_i}\right] = \sum_i v_i \mathbb{E}\left[e^{\sigma \sum_{j \leq i} \tilde{\Delta}_i}\right].$$

Given all of the layers of indirection here (that $\tilde{\Delta}_i$ depends on S_i and S_{i-1}, which depend on F_i, F_{i-1}, which depend on v_i and v_{i-1} (as well as the time-of-day feature for the interval corresponding to index i), this formula is still quite complicated and difficult to work with. In addition, the quality of our feature selection and modeling in this rather noisy environment is likely too poor to support a highly nuanced form for all of these related probability distributions. As a result, we will employ a simplifying heuristic and treat the relationships between v_i and F_i as well as the relationships between S_i, S_{i-1} and $\tilde{\Delta}_i$ as if they were deterministic, always taking on the values of their expectations. [Note: this is a highly suspect thing to do, and we feel this is another prime target for improvement as we continue our research and build up better feature selection and modeling.]

Under this heuristic, we can rewrite our cost estimate as:

$$C(v_1, \ldots, v_N) := \sum_i v_i e^{\sigma \sum_{j \leq i} \mathbb{E}[\tilde{\Delta}_i]}$$

Let's consider this term $\mathbb{E}[\tilde{\Delta}_i]$ in isolation for a moment. We'll treat the time-of-day feature as having a fixed value t_i. If we let $\mathcal{S}$ represent the set of all possible market states with $T_i = t_i$ (i.e. iterating over all possible pairs of values for our pressure gauge features), we can write this as:

$$\mathbb{E}[\tilde{\Delta}_i] = \sum_{m_i, m_{i-1} \in \mathcal{S}} \mathbb{P}(m_i | t_i) \mathbb{P}(m_{i-1} | t_i) \mathbb{E}[\tilde{\Delta}_i | s_i, s_{i-1}, t_i],$$

where $s_i := m_i + f_i$ and $s_{i-1} := m_{i-1} + f_{i-1}$. Under our heuristic that treats f_i and f_{i-1} as deterministic functions of v_i, v_{i-1}, we can estimate this quantity from recent historical market data as follows. For each symbol and pair of adjacent time intervals in our data set, we compute the notional value traded in each interval and take the product of the two. This represents a weight that is proportional to the probability of this observation as a sample for (m_{i-1}, m_i) under the joint (independent) probability distribution we've defined (conditioned on t_i). We also compute the state values for each of the adjacent intervals, and the $\tilde{\Delta}$ value. Next we can aggregate all of the observations that share the same pair of adjacent state values and compute the average $\tilde{\Delta}$ value, under the weights of our joint probability distribution. For each pair of state values (s_{i-1}, s_i), we'll let W_{s_{i-1},s_i} denote the sum of all the weights of observations with these state values. We'll let $\tilde{\Delta}_{s_{i-1},s_i}$ denote the weighted average of the $\tilde{\Delta}$ observations with these state values (relative to these same joint distribution weights). [Technical note: there is one nuance here that we are being cavalier about - the value of t_{i-1} is part of the state for interval $i-1$, so terms like $\mathbb{P}(m_{i-1}|t_i)$ should really be conditioned on both t_i and t_{i-1}. However, tracking and conditioning on t_{i-1} as well as t_i would further split our data and increase our risk of over-fitting. Given how weak the time-of-day feature seems to be in terms of its predictive power, we choose not to do this further conditioning.]

We note that W_{m_{i-1},m_i,t_i} is proportional to the $\mathbb{P}(m_i|t_i)\mathbb{P}(m_{i-1}|t_i)$, and $\tilde{\Delta}_{s_i,s_{i-1}} = \mathbb{E}[\tilde{\Delta}_i|s_i, s_{i-1}]$. We then have:

$$\mathbb{E}[\tilde{\Delta}_i] = \frac{\sum_{m_i,m_{i-1}\in\mathcal{S}} W_{m_{i-1},m_i,t_i}\tilde{\Delta}_{m_{i-1}+f_{i-1},m_i+f_i,t_i}}{\sum_{m_i,m_{i-1}\in\mathcal{S}} W_{m_{i-1},m_i,t_i}}.$$

This is for fixed values of f_{i-1} and f_i, so we could think this as a function of f_i and f_{i-1}. In turn, we are heuristically viewing each f_i as a deterministic function of v_i for now, which we'll denote by $f_i := f(v_i)$. Putting this together, we can think of and denote this expectation as a function of v_i,v_{i-1}, and t_i:

$$\Delta(v_i, v_{i-1}, t_i) := \frac{\sum_{m_i,m_{i-1}\in\mathcal{S}} W_{m_{i-1},m_i,t_i}\tilde{\Delta}_{m_{i-1}+f(v_{i-1}),m_i+f(v_i),t_i}}{\sum_{m_i,m_{i-1}\in\mathcal{S}} W_{m_{i-1},m_i,t_i}}.$$

We can plug this representation into our overall cost function:

$$C(v_1, \dots, v_N) := \sum_i v_i e^{\sigma\sum_{j\le i}\mathbb{E}[\tilde{\Delta}_i]} = \sum_i v_i e^{\sigma\sum_{j\le i}\Delta(v_j,v_{j-1},t_j)}. \quad (1)$$

This gives us a cost function that we can compute for any specified values of σ and $v_1, \dots, v_N$. Computing the $\Delta(v_i, v_{i-1}, t_i)$ values from historical data is a time-intensive process, but we can collect and aggregate data over a few months of data and compute these for all reasonable v_i and v_{i-1} within a day or so. It is important to remember that we've employed many heuristics here as well as empirical estimates. In particular, we should not expect these estimates to behave well if we attempt to do this for rather unusual/extreme values.For modest values of the v_i's, however, we can perform sanity checks to see that these estimates behave reasonably and match with our general intuition.

So what should we generally expect from $\Delta(v_i, v_{i-1}, t_i)$ as a function? Well, it's easier to reason about if we fix two of the three input values at a time and look at the behavior as the one remaining input varies. Here are the three intuitive behaviors we might guess we will/should see in those cases (all assuming that our volume v_i is on the side of buying. Things are analogous with reversed signs for the case when we are selling).

1. If we fix v_{i-1} and t_i and vary v_i, Δ will be an increasing function of v_i.
2. If we fix v_i and t_i and vary v_{i-1}, Δ will be a decreasing function of v_{i-1}.
3. If we fix v_i and v_{i-1}, Δ will be largest when $t_i = 0$ and smallest when $t_i = 1$.

The first of these guesses comes from our basic hypotheses of how our pressure gauges will behave. More buying "pressure" in the current interval should lead to an increasing price, while more selling pressure should lead to a decreasing price. The second comes from our intuitive sense of reversion: some expectations of the market based on the prior interval may be baked in, so the response to the behavior in the current interval will be filtered through that lens. An interval with a certain fixed amount of pressure is likely to have more impact when it follows a lower pressure interval than when it follows a higher pressure one (or at least, that's our hypothesis). The third item above comes from our intuition that trading is most volatile in the morning, so impact may be heightened then, while impact is likely to be dampened in the middle of the day and then perhaps pick up a little again heading into the close.

We further have expectations about the relationship between v_i values and f_i values (which we are treating as deterministic by computing its average and using solely that). We expect that increases in v_i when we are buying, for example, will lead to increases in the pressure gauges on average (in the direction that pushes prices up). [Aside: this expectation holds up on our preliminary experimental data.] So in essence item 1. above is proposing: if we fix the value of the t_i and the f_{i-1} values for the previous interval, as well as one of the two pressure components of f_i for the current interval, we expect an increasing function as we increase the other component of f_i in the current interval i (which is the only remaining non-fixed variable).

We won't detail every example of possible fixed values here as that would quickly become unwieldy, but we'll spot check a few and then attempt to give a holistic sense of how true these three suppositions are for our empirically estimated Δ function.

For a quick spot check on item 1., let's start with the example where $t_i = 0$, both components of f_{i-1} for interval $i-1$ are set to 0, and the volume pressure gauge component of f_i for interval i is set to 0. As the join pressure gauge component of f_i for interval i increases from 0 to 4, we get the following values for Δ (in order and rounded to the nearest two decimal points): $0, 0.38, 0.46, 0.62, 0.92$. These values are increasing, as we expected. If we do the same thing but now fix the current join pressure component of f_i to 0 and let the current interval's volume pressure component go from 0 to 4, we get the following values for Δ: $0, 0.38, 0.63, 0.90, 1.27$. These are again increasing.

For a quick spot check on item 2., let's look at the example where $t_i = 1$, both pressure gauge components of f_i in the current interval are set to 0, and the volume pressure gauge component of f_{i-1} for interval $i-1$ is also set to 0. As the join pressure gauge component of f_{i-1} increases from 0 to 4, we get the following values for Δ: $0, -0.14, -0.15, -0.16, -0.29$, which are decreasing, as expected. Doing the same thing but setting the prior join pressure component to 0 and varying the volume pressure component of f_{i-1}, we get Δ values of $0, -0.08, -0.12, -0.22, -0.27$.

For a quick spot check on item 3., let's look at the example where all of the pressure gauge components of f_i and f_{i-1} are set to 1, and the time of day feature varies from 0 to 2. In this case, we get Δ values of: $0.32, 0.19, 0.25$. This conforms to the pattern we expect. (Note that the example of all pressure gauge components of f_i and f_{i-1} being set to 0 just shows flat 0 Δ values despite variation in t_i, so this is uninteresting.)

It's not too hard, however, to find examples where our empirical results do not conform exactly to these expected patterns. For example, if we vary the time-of-day feature again while setting almost all of the pressure gauge components to 1 but setting the current interval's volume pressure gauge component to 2, we get Δ values of: $0.51, 0.44, 0.28$. So the trend of the morning's value being largest still holds, but now the late day number is less than the mid-day one. (Note that as would we expect, these impact estimates are larger than the estimates when all of the pressure gauge components of f_i and f_{i-1} were set to 1.)

There are lots of possibilities for how we can fix a subset of the variables and study the behavior of this Δ function. For now, what we'll do is iterate through the various fixed settings for each of the cases 1 through 3 above and compute the ordering of the Δ values as the remaining variable varies, from lowest to highest. For example, a strictly increasing function corresponds to the ordering 1,2,3, while a strictly decreasing function corresponds to the ordering 3, 2, 1. We'll then count how many occurrences of each ordering we see as we iterate through the possible fixed values of the other variables.

We'll start with looking at cases where the volume pressure component of f_i varies, with all other values held constant. Iterating over the all combinations of the three values for the time-of-day feature and values 0,1,2,3,4 for the other pressure gauge components of f_i and f_{i-1}, we are examining a total of 375 functions of the volume pressure component of f_i. Of these, 297 are strictly increasing (this is nearly 80% of them). The vast majority of the remainder display one transposition in the ordering: e.g. the ordering is something like 0,2,1,3,4 instead of 0,1,2,3,4, so there is just one switch of two adjacent values as compared to the expected ordering. Only 16 of the total 375 orderings display more than one transposition as compared to the default ordering of 0,1,2,3,4.

We can do the analogous computation for what happens as the join pressure component of f_i varies. In this case, 222 of the 375 orderings match the default ordering of 0,1,2,3,4 for a strictly increasing function. This is about 60% of them. If we include orderings just one transposition away from this default, we cover 314 of the 375 orderings.

Next we do the same computation for what happens as the volume pressure component of f_{i-1} varies. Here we expect to see a lot of the opposite ordering, 4,3,2,1,0, corresponding to a strictly decreasing function due to reversion. The concentration on this ordering is weaker, but that is mostly what we see. Of the 375 orderings, 105 of them match 4,3,2,1,0, which is by far the mode. If we include single transpositions away from this ordering, we cover a total of 200 of the 375 orderings (about 53%).

Doing the same analyses for the join pressure component of f_{i-1}, we get weaker but still reasonable results. The ordering of 4,3,2,1,0 remains the clear mode, occurring 56 times out of 375. If we include single transpositions, we cover 142 of the 375 orderings.

For the time feature varying, we are iterating over five values for each of the four pressure gauge component variables, which means we are examining 625 orderings. For these, the standout mode is the strictly decreasing ordering 2,1,0, which occurs 440 times. The ordering of 1,2,0 (corresponding to our hypothesis that the morning would have the highest impact estimates and the middle of the day the least) is the next most common, occurring 92 times.

We probably shouldn't read too much into any of these numbers, as this is kind of crude and ad-hoc way to gain some holistic understanding of our empirical results for Δ. In particular, we have not weighted the relative importance here for each fixing of variables by how commonly those fixed values occur, so we are getting more an overview of the landscape of possible behaviors, rather than a clear picture of *likely* behavior. There are lots of other computations we can and should do here, but as a basic sanity check to see if our empirical results seem reasonable, we feel this passes for now.

The time-of-day feature results in particular suggest some promising avenues for further investigation. It would be interesting to try to understand more systemically under what conditions we expect impact to be lesser in the late day vs. mid-day time periods and why.

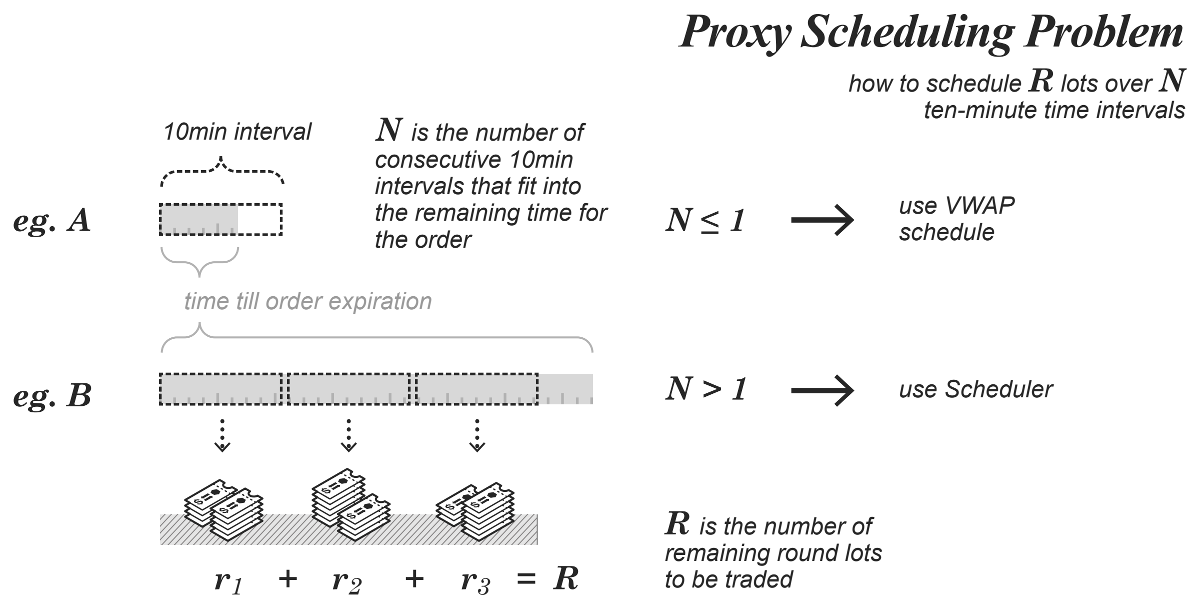
For now though, we are satisfied that the cost function in (1) provides a reasonable (though still very noisy!) proxy for comparing the expected impact costs of different possible schedules, and so we can use this as the basis for the design of a scheduler that tries to minimize our expected impact costs. In the next section, we'll detail our scheduler design.

7 A Scheduler Using Dynamic Programming

As a component of our algo logic, a scheduler should be designed to give sensible answers to any question of the form, "how many shares of this order should we trade over the next X minutes?" We are going to design here a scheduler for intra-day trading, and leave allocations to auctions as a separate task. In our implementation of this algo, the amount allocated to opening and closing auctions is chosen proportionately to the historical volume curve, and the remaining volume is allocated via the intra-day scheduler we are discussing here.

We should not assume ahead of time that our intra-day scheduler will be invoked only at particular times or for particular values of X. In fact, our algo tactics involve a fair amount of randomization, with the result being that there is no dependable structure to the scheduling questions the scheduler is asked to solve. As we have seen, however, data science in highly noisy data sets is precarious, and we would probably not want to build multiple models or modeling frameworks for a wide variety of similar time scales. Instead, we want to reasonably adapt the outputs of the models we have built at the timescale of 10-minute intervals to provide answers for any reasonable scheduling problem, even if it does not cover the full trading day and does not align with 10-minute boundaries, etc.

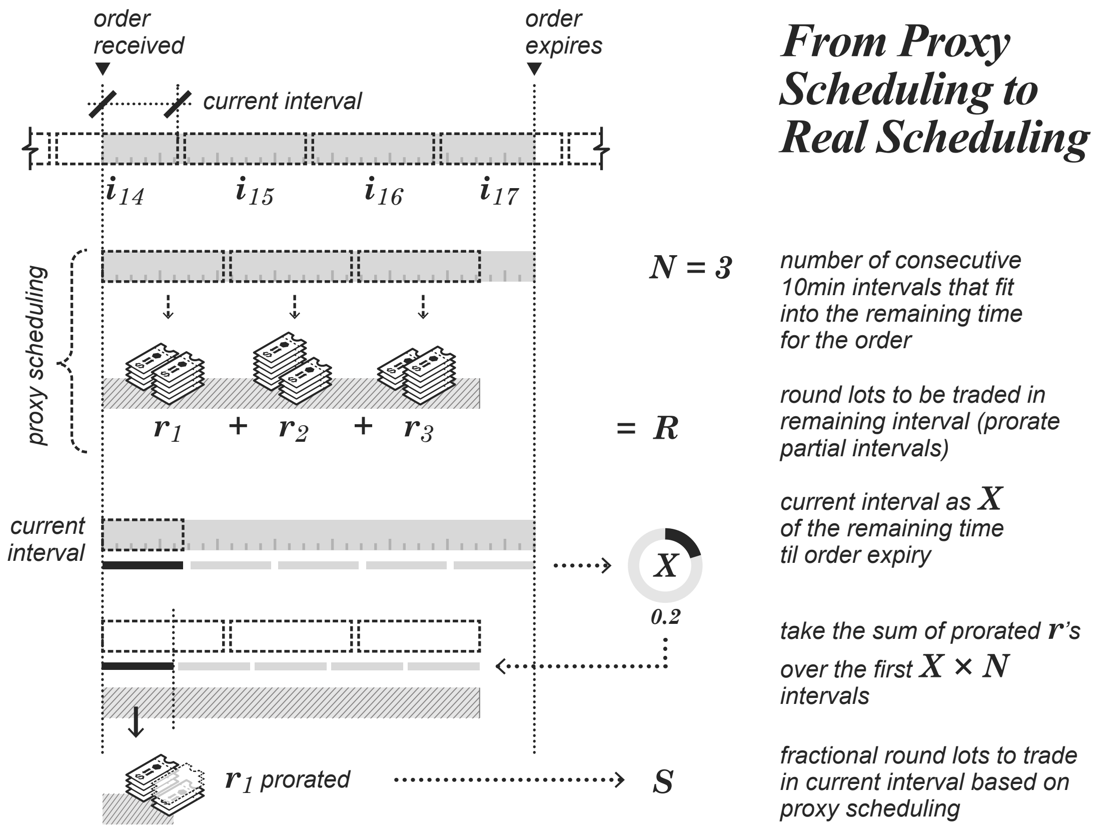
When asked to determine how much to schedule for the next time interval, our new scheduler will consider several parameters. The most notable are: the current time, the order expiration time, how many shares are left to trade, and the endpoint of this newly beginning time interval that we are being asked to schedule for. When the remaining time to order expiration is less than ten minutes, we'll default to a VWAP schedule, since this is a shorter total time horizon than what our price impact research is really designed to inform. When the total time remaining for the order is greater than ten minutes, we'll first compute how many consecutive ten minute intervals would fit approximately into the remaining time. Let's call that number N. We'll also compute how many round lots are remaining to be traded, and we'll call that number R.


Proxy Scheduling Problem
how to schedule R lots over N ten-minute time intervals
10min interval
N is the number of consecutive 10min intervals that fit into the remaining time for the order
eg. A
N ≤ 1 → use VWAP schedule
time till order expiration
eg. B
N > 1 → use Scheduler
R is the number of remaining round lots to be traded
r1 + r2 + r3 = R

Now we can define a scheduling problem that is closely related to the task at hand, though not exactly the same: what do we think is the best way to schedule the trading of

R lots over N ten-minute time intervals? We'll call this our "proxy scheduling problem." Skipping ahead for a moment, suppose we had an answer to this proxy scheduling problem in the form: "first trade r_1 round lots in the first interval, then r_2 round lots in the second interval, ..., and finally trade r_N rounds lots in the last interval," where $r_1 + \cdots + r_N = R$. With this kind of solution in hand, we could reasonably answer scheduling questions about time intervals that didn't match up perfectly with our ten-minute intervals by pro-rating our r_i quantities for any partial intervals. For example, if asked how much we would suggest trading over the next five minutes, we would say $0.5 * r_1$. If asked how much we would suggest trading over the next twenty-three minutes, we would say $r_1 + r_2 + 0.3 * r_3$. [Technical note: we probably want to round these answers back to an integer number of round lots in most cases.]

But how do we handle the further discrepancies between this clean "R lots in N 10-minute intervals" proxy scheduling problem and our messier real scheduling problem? The number of shares remaining might not be an exact multiple of round lots, and the current time and order expiration times might not line up with 10-minute boundaries. To deal with such issues, we'll think in ratios that can be translated between the real problem and the proxy problem. More precisely, we'll compute: in the real scheduling problem, the current interval to be scheduled represents X of the remaining time until order expiry, where X is a number between 0 and 1. We can then apply this X to determine how we want to pro-rate the r's we obtain as the solution to our proxy scheduling problem. For example, if X is 0.14, then we want to take the sum of the (pro-rated) r's over the first $0.14 * N$ intervals. Let's call that resulting sum S. Now S represents a fraction $Y := S/R$ of the remaining volume in the proxy scheduling problem. We can then take Y and apply it to our real scheduling problem: we'll multiply Y by the remaining shares left to be traded to determine an answer to our real scheduling query. [Technical note: our algo in production will typically round this answer to an integer number of round lots, and also will apply some randomization that may make the final amount scheduled a bit smaller or larger than this initially computed answer.]


order received
order expires
current interval
i_{14} i_{15} i_{16} i_{17}
proxy scheduling
From Proxy Scheduling to Real Scheduling
N = 3
number of consecutive 10min intervals that fit into the remaining time for the order
r_1 + r_2 + r_3 = R
round lots to be traded in remaining interval (prorate partial intervals)
current interval
current interval as X of the remaining time til order expiry
X
0.2
take the sum of prorated r's over the first X × N intervals
r_1 prorated
S
fractional round lots to trade in current interval based on proxy scheduling

By this point, you may rightly be wondering: why have we worked so hard to define and translate back and forth to this proxy scheduling problem? It's really the same kind of problem we started with, so this doesn't feel like forward progress. But as it turns out, the proxy problem is very conveniently designed to be solved with dynamic programming.

Let's consider what all of our work in the previous sections of this paper has yielded: a way to compute a cost function for trading a sequence of proscribed amounts (e.g. $r_1, \ldots, r_N$) over 10-minute intervals, as long as those amounts stay within specified caps. As we've discussed already above, however, we can't afford the time and computational resources it takes to compute the expected cost estimates directly for *all* of the reasonable schedules we might consider, so this is where dynamic programming techniques comes in handy.

Our task is to find the schedule with the lowest cost estimate among the set of possibilities under consideration for our proxy scheduling problem. We'll do this by considering the 10-minute intervals one at a time, working backwards from the end. For ease of description here, we're going to temporarily ignore some nuances, and in a bit we'll discuss how to work them back in. In particular, we're going to ignore that our cost estimates can depend upon our time of day feature.

First we'll notice that we can extend the definition of our cost function C from (1) to arbitrary suffixes of volume sequences:

$$C(v_{k-1}, \dots, v_N) := \sum_{i=k}^{N} v_i e^{\sigma \sum_{k \leq j \leq i} \Delta(v_{k-1}, v_k)}.$$

This extended definition computes a value that is proportional to our cost estimate for the last k intervals, though it also must include v_{k-1} in the input to determine the first Δ term. We say the value is proportional because we have omitted the factor of $e^{\sigma \sum_{j<k} \Delta(v_{k-1,k})}$ that gets multiplied by this in our cost function when we consider the full sequence $v_1, \dots, v_N$. We define $v_0 := 0$ for notational consistency.

Our extended cost function definition satisfies a recursive property:

$$C(v_{k-1}, \dots, v_N) = e^{\sigma \Delta(v_{k-1}, v_k)} \left(v_k + C(v_k, \dots, v_N)\right).$$

We can also define an "optimal" cost function C^* that takes v_{k-1}, V, N, and k as input:

$$C^*(v_{k-1}, V, N, k) := min_{\substack{v_k, \dots, v_N \text{ s.t.} \\ v_k + \cdots + v_N = V}} \{C(v_{k-1}, \dots, v_N)\}.$$

Here, the minimum of C is taken over all valid volume sequences $v_k, \dots, v_N$ that add up to the specified total amount V.

This minimized cost function C^* also satisfies a recursive property:

$$C^*(v_{k-1}, V, N, k) = min_{v_k} e^{\sigma \Delta(v_{k-1}, v_k)} \left(v_k + C^*(v_k, V - v_k, N, k+1)\right) \tag{2}$$

Here, the minimum of C^* is taken over valid values of v_k, which must be non-negative and $\leq V$. We may impose a tighter, absolute cap on individual v_i values as well.

Now we can consider our proxy scheduling problem backwards, starting with possibilities for the last interval and iteratively extending our view. As we go, we'll want to keep track of what we've learned about the possible paths ahead of us. We'll suggestively conflate notation here and treat C^* as a data structure whose entries are indexed by values of v_{k-1}, V, N, and k. [Technical note: we can treat the parameter N as a constant throughout, so actually our data structure has a 3-dimensional index. But we'll keep writing N because as a parameter here because we feel it is clearer for exposition.] We initialize our data structure with values of $+\infty$, meaning that cost estimates are thought of infinite until proven otherwise.

When $k = N$, there is only one way to choose a $v_k = v_N$ such that $\sum_{N \leq j \leq N} v_j = V$: namely, we must set $v_N = V$. Thus, for every V that we view as a valid amount to schedule in the final interval and every valid possible value of v_{k-1}, we can overwrite the $+\infty$ cost entry for $C^*(v_{k-1}, V, N, N)$ with:

$$C^*(v_{k-1}, V, N, N) = e^{\sigma \Delta(v_{k-1}, V)} * V.$$

There is something important to note about the structure of equation (2). All of the referenced values of C^* on the right hand side of the equation are for a strictly higher value of the last parameter, $k \to k+1$. Thus, if we assume we have already computed all of the values of C^* for valid combinations of v_k and V with our same fixed value of N and for some value $k+1$, then we can compute all of the relevant values for k from these, using the specification in equation (2). We simply cycle through all of the possible values for v_k, and take the minimum value of $e^{\sigma\Delta(v_{k-1},v_k)}\,(v_k + C^*(v_k, V - v_k, N, k+1))$ that we find, and we store this in our data structure as the value of $C^*(v_{k-1}, V, N, k)$. Having set all of the entries for $k = N$ correctly, we can thus work backwards, next setting all of the entries for $k = N-1$, and then all of the entries for $k = N-2$, and so on.

This is the general form of dynamic programming: we take a parameterized problem we want to solve, we find a recursive relationship between it and its natural subproblems, and we identify an order for solving the subproblems such that each time we solve a new problem, we can use the recursive relationship and it only references subproblems we've already covered.

The remaining question is: how much is this all going to cost us in terms of computational time and memory? For this, we need to quantify how many valid values of v_{k-1} and V we may need to consider. To control these, we can place a cap on how much volume can be scheduled in any single interval, and we can also discretize our choices for v_k's to whatever level of coarseness we wish. Let's let A denote the number of discrete values for v_{k-1} that we consider valid for a particular interval. (For simplicity for now, we'll assume this number is the same for every interval, but that isn't a requirement in general.) The number of values of V that can be achieved is then $\leq N * A$, and the values of k range from 1 to N. Thus, an upper bound on the number of values for C^* that we will need to store is: $A^2 * N^2$.

Each evaluation of equation (2) involves finding the minimum among $\leq A$ pre-computed values, so our total computation time can be upper bounded as: $\mathcal{O}(A^3N^2)$. By controlling how many choices we consider for the scheduling in each individual interval (i.e. the value of A), we can control this value of A^3N^2 and keep it in a range that is reasonable for our algorithm to compute in real-time when faced with a scheduling problem.

One fun aside before we leave the topic of dynamic programming: if you're thinking that the term "dynamic programming" seems engineered to sound vaguely impressive while not conveying much, you're right! It was coined by Richard Bellman in the 1940s for essentially that purpose. As Bellman writes in "Eye of the Hurricane: An Autobiography" (1984, p. 159):

"I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, "Where did the name, dynamic programming, come from?" The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word "research." I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word "programming." I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying. I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities."

7.1 A Few Illustrative Examples

Now that we've gone through all the work to set ourselves up to compute schedules that minimize our estimates of impact cost (subject to trading the right amount of volume), we may ask: what will these schedules look like? Will all this work lead to trivial answers that just spread the volume out evenly or just concentrate it as heavily as possible? The answer, it turns out, depends heavily on the nature of the function $\Delta(v_i, v_{i-1})$.

To gain some intuition about how the Δ function drives the scheduling behavior, we can consider some toy examples. For this we'll substitute very simple functions for Δ and then further simplify with heuristic estimates of equation (1) that allow us to solve for an "optimal" schedule directly. These examples should not be expected to represent the real behavior of our algorithm in practice, as all of these simplifications are unrealistic and chosen for convenience rather than faithfulness to real data. However, they help us gain an understanding for the possible *range* of scheduling behaviors that our framework is capable of introducing, as well as understanding of how properties of Δ functions can drive corresponding behaviors in the resulting schedules.

We'll start with a toy example that exhibits no dependence on v_{i-1} (for one thing, this means no reversion). We define:

$$\Delta(v_i, v_{i-1}) := \alpha v_i,$$

for some positive constant α. In this case, we can rewrite equation (1) as:

$$C(v_1, \ldots, v_N) = \sum_i v_i e^{\sigma \sum_{j \le i} \mathbb{E}[\tilde{\Delta}_i]} = \sum_i v_i e^{\sigma \sum_{j \le i} \alpha v_j}.$$

By redefining α, we can absorb σ and write this as:

$$\sum_i v_i e^{\alpha \sum_{j \le i} v_j}.$$

Next, we'll make a simplifying approximation by replacing e^x with $1+x$, the beginning of its Taylor series expansion at $x=0$. This gives us:

$$\sum_i v_i(1+\alpha\sum_{j\le i} v_j) = \left(\sum_i v_i\right) + \alpha\left(\sum_{\substack{i,j\\ j\le i}} v_i v_j\right).$$

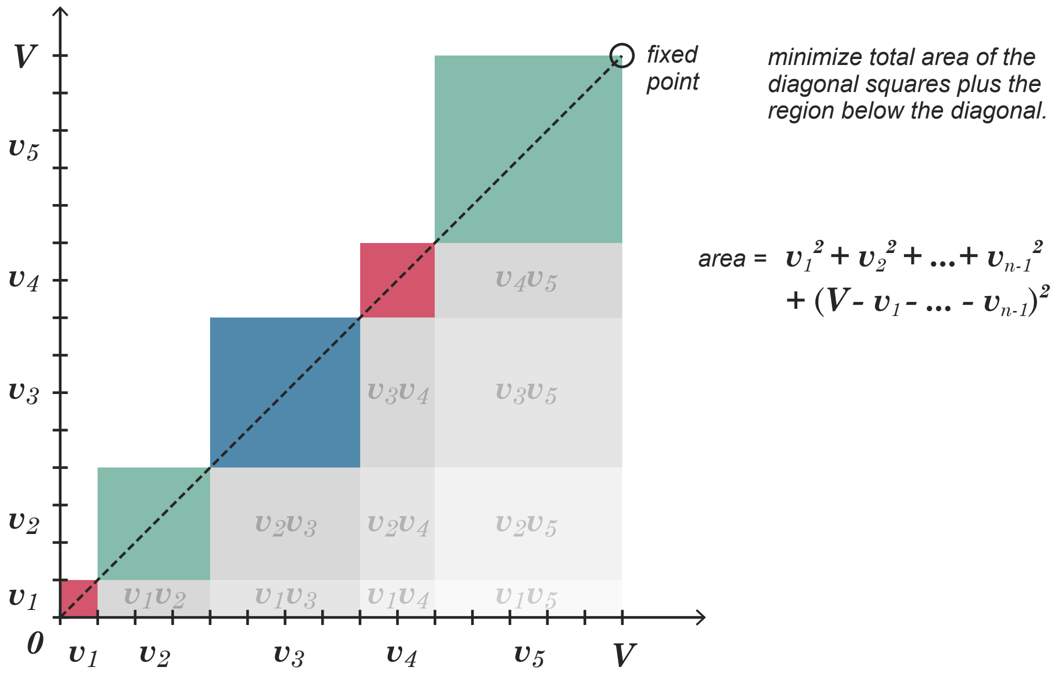
This first term, $\sum_i v_i$, is just the total volume to trade. Since we are treating that as a fixed constant V, we can ignore that terms for the purposes of minimization. Thus, our goal is to minimize

$$\alpha\left(\sum_{\substack{i,j\\ j\le i}} v_i v_j\right),$$

subject to the constraints each of our N variables v_i must satisfy $0 \le v_i \le V$ and $\sum_i v_i = V$.

This is an example of what is known as a "quadratic programming problem." There is a fair amount of optimization theory devoted to the solving of quadratic programming problems: the general case is doable, well-studied, and somewhat complicated. It so happens that this particular example has a straightforward solution.

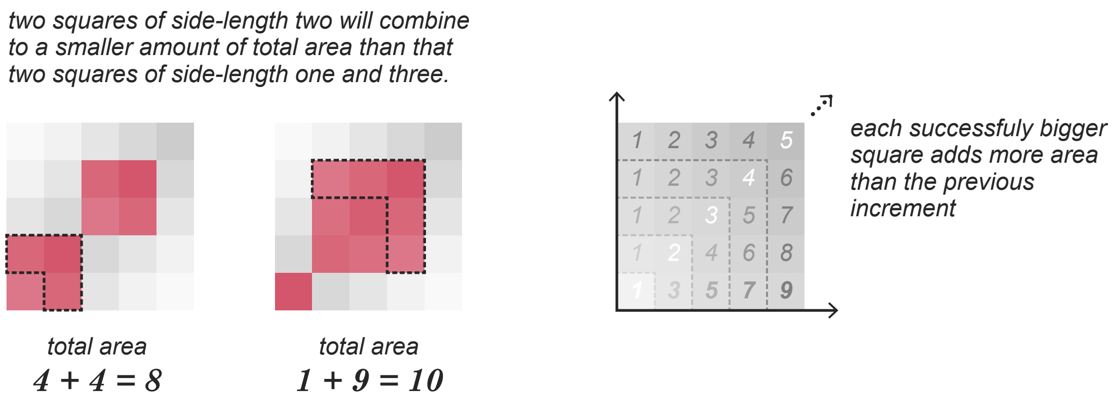
We can decompose this value $\sum_{\substack{i,j\\ j\le i}} v_i v_j$ into a sum of diagonal terms, $\sum_i v_i^2$, and a sum of terms below the diagonal, $\sum_{\substack{i,j\\ j<i}} v_i v_j$. This is probably clearest if we visualize things geometrically. The quantity v_i^2, for instance, has a geometric meaning as the area of square with side-length v_i. And each quantity $v_i v_j$ has a geometric meaning as the area of a rectangle with length v_i and height v_j. If we draw a 2-dimensional grid, we can think of all these shapes as living inside the square with corners $(0,0)$, $(0,V)$, $(V,0)$, (V,V). Each axis from 0 to V can be divided into consecutive regions of length $v_1, v_2, \ldots, v_N$, and then all of the points (x,y) where x falls inside the v_i region and y falls inside the v_j region for a rectangle with area $v_i v_j$. The squares corresponding to the v_i^2 values envelop the diagonal line that connects $(0,0)$ to (V,V):



fixed point
minimize total area of the diagonal squares plus the region below the diagonal.
area = $v_1^2 + v_2^2 + \ldots + v_{n-1}^2 + (V - v_1 - \ldots - v_{n-1})^2$
V
v_5
v_4
v_3
v_2
v_1
0
v_4v_5
v_3v_4
v_3v_5
v_2v_3
v_2v_4
v_2v_5
v_1v_2
v_1v_3
v_1v_4
v_1v_5
v_1 v_2 v_3 v_4 v_5 V

Assuming $\alpha > 0$, we want to choose the values of $v_1, \ldots, v_N$ in such a way that minimizes the total area of the diagonal squares plus the region below the diagonal. The region below the diagonal gets fully covered regardless, so this boils down to minimizing the over-diagonal contributions of the squares that sit along the diagonal.

Let's think about the nature of these squares. If you look at the possible square sizes as nested layers, it's easy to see that each successively bigger square adds more area than the previous increment. This means, for instance, that two squares of side-length two will combine to a smaller amount of total area than a square of side-length one and a square of side-length three. To see this, we can imagine starting with the two squares of side length two, and imagining that one sheds a layer and gives it to the other. The shedded layer (the increment from a square of side-length one to a square of side-length two) has area $= 3$. But the added layer (the increment from a side-length two to a square of side-length three) has area $= 5$.


two squares of side-length two will combine to a smaller amount of total area than that two squares of side-length one and three.
total area
4 + 4 = 8
total area
1 + 9 = 10
1 2 3 4 5
1 2 3 4 6
1 2 3 5 7
1 2 4 6 8
1 3 5 7 9
each successfuly bigger square adds more area than the previous increment

Fixing the sum of $v_1, \ldots, v_N$ means we are fixing the sum of the side-lengths of all of the squares. This is like a fixed number of layers overall. If we were to start with some arbitrary values for $v_1, \ldots, v_N$, we know that taking layers from larger v_i's and giving them to smaller v_i's instead will reduce the total area. This means that the minimal solution will have v_i values that are as equal as possible. Thus, this toy example leads to a TWAP-like schedule, where we would schedule roughly equal amounts of volume for each time interval.

Next let's consider a toy example that exhibits only dependence on v_{i-1} (this represents a delayed reaction). We define:

$$\Delta(v_i, v_{i-1}) := \alpha v_{i-1},$$

for some positive constant α. In this case, we can rewrite equation (1) as:

$$C(v_1, \ldots, v_N) = \sum_i v_i e^{\sigma \sum_{j \le i} \mathbb{E}[\tilde{\Delta}_i]} = \sum_i v_i e^{\sigma \sum_{j<i} \alpha v_j}.$$

We will again simplify using the approximation $e^x \approx 1 + x$ for small values of x and absorb σ and α into a single constant:

$$\alpha \sum_i v_i (1 + \sum_{j<i} v_j).$$

The relevant term to minimize here is $\sum_{\substack{i,j \\ j<i}} v_i v_j$. This should be familar now as the area of the under-diagonal rectangles we illustrated in the example above. This time though, because we looking at a delayed reaction case, there are no diagonal terms included. As a result, we should look to *maximize* the total area of the missing diagonal terms, in order to take a bigger bite out of the under-diagonal space we are trying to minimize. This should lead us to concentrate the volume as much as possible, making some v_i as large as we can and then putting as much as the remaining flow as we can into another v_j and so on. This kind of schedule will be the opposite of a TWAP and will try to trade as chunk-ily as possible. We note that for this particular toy version of the optimization problem, it doesn't ultimately

matter what order the v_i values go in. This is easy to see from the geometric view - if we reorder the square sizes along the diagonal, the total area that gets "cut out" from under the diagonal by their shapes doesn't change.

We can generalize these two toy cases to a form of:

$$\Delta(v_i, v_{i-1}) := \alpha v_i + \beta v_{i-1}$$

for some constants α and β. Applying the same approximation heuristics we used above, we would get a simplified optimization problem requiring us to minimize something of the form:

$$a\left(\sum_i v_i^2\right) + b\left(\sum_{\substack{i,j\\ j<i}}\right),$$

where a and b are constants. If $b \leq a$, it is clear that the minimum is achieved by minimizing the area of the diagonal squares, and hence we will want to spread volume out as evenly as possible. If we fix some value of a and then begin increasing b, at some point where $b > a$, there will start to be an incentive to carve out more of the under diagonal space with the now cheaper diagonal squares, eventually leading to volume that is as concentrated as possible.

All of this is just a small taste of the kinds of optimization problems that can arise as the details of the Δ function vary, and as the form of our cost function C or our approximation of it vary. Terms like v_i^2 may act to push the optimal answer towards evenly spread volume, while terms like $v_i v_j$ for $j \neq i$ may act to push the optimal answer towards concentrated volume. Other kinds of terms may push the optimal answer towards spurts of concentrated volume followed by cooling off periods. For example, imagine we are trying to minimize an expression like:

$$\sum_{\substack{i,j\\ j\leq i}} v_i v_j v_{j-1}. \tag{3}$$

This could arise as part of our cost expression if we had something like $\Delta(v_i, v_{i-1}) := \alpha v_i + \beta v_{i-1} + \gamma v_i v_{i-1}$. The $v_j v_{j-1}$ multiplication in expression (3) ensures that the contribution from this term will be 0 if we can allocate volume only in every other interval.

Overall, the schedule that minimizes our expected costs while respecting all of our constraints arises from a tug-of-war between the component pieces of our cost function. Any functional forms we could choose to impose (like fitting $\Delta(v_i, v_{i-1})$ as linear, for example) will have a strong or even determining influence on the outcome. For now, we try to avoid assuming a specific form for Δ and instead rely upon our empirical estimation of $\Delta(v_i, v_{i-1})$ for relevant values of v_i and v_{i-1}. Finding a "good" functional form for Δ is a compelling goal for our future research, as that would allow us to try to solve the resulting optimization problem more directly, perhaps leading to a simpler and/or faster solution method compared to dynamic programming. It could also give us more insight into market dynamics, and inspire a better form for our cost function or other aspects of this framework.

8 Summary

In this work, we have detailed the research underpinning the design of the scheduler for our newest algo, a scheduler which is intended to minimize impact conditioned on completing the volume it is assigned within the allotted time. This is just one piece of our new algo, which also includes a liquidity seeking component that we will discuss elsewhere.

At this point, the features we use for our modeling as well as the models themselves are fairly basic, as fitting complex models to extremely noisy data is very fraught, and we preferred to start with something more basic and hopefully more robust. We intend to iterate quickly and frequently on this research and all other aspect of our design, as we study our algos behaviors in practice and do more research on historical market data sets. We will continue to release new findings as they develop.

9 Acknowledgements

We would like to thank everyone who consulted with us on earlier versions of this work and this draft. We would especially like to thank our former quantitative researcher Matthew Schoenbauer, who contributed to this project in its earlier stages.

A Some Dirty Laundry

The precursor to this research began in the fall of 2019, when we designed a pretrade impact model. That work involved early analogs of the work we've done here in sections 4 and 6. We've been working on these topics steadily ever since. The path we took to get to our current state of understanding was a winding one, littered with software bugs, sub-optimal decision making, and dead ends. We expect the path forward from here will be similar, and some day in the not-to-distant future we will likely look back upon this checkpoint of our work as embarrassingly primitive. Such is the nature of research, if we are lucky.

Feature selection in particular has been a bumpy road. Some seemingly natural ideas we tested for feature selection did not pan out, and we'll share one of those here as an illustrative example. Intuitively, we expected that if we fixed an amount of volume that we intended to trade in a given interval, the impact of that fixed amount would be smaller in times of heavier overall market volume. This is like expecting the sound of penny dropping to garner more reaction in a quiet room than in a loud room. If we could reliably detect and measure such an effect, we could incorporate volume curves or real-time volume information or similar into our modeling process to account for it.

There are many ways to go about looking for such an effect, and we tried a few of them so far, to no avail. One of our first ideas was to include a feature computed by taking the total volume in the previous 10-minute interval and dividing by the symbol ADV. It was our

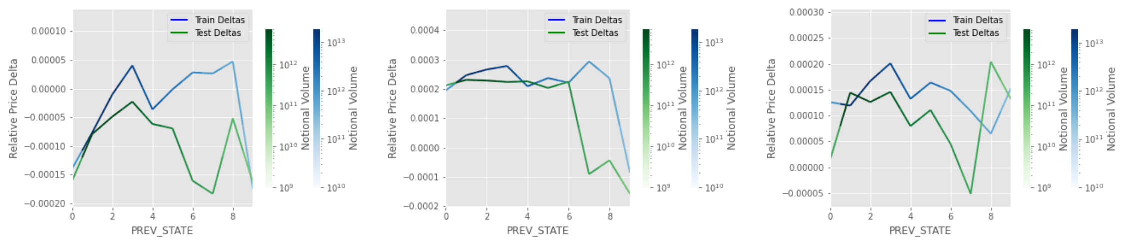


Figure 1: Our first previous volume feature attempt: Not a success

intuition that a high amount of activity in the previous interval would allow our activity to have less of an effect on price in the current interval.

Our first attempt to see if this feature might meaningfully help was to look at plots of how average price movement varied as a function of this feature, while holding constant the amounts of volume that we intended to add to the current and previous intervals. In other words, we did something similar to what we did in section 6 above to get a look at $\Delta(v_i, v_{i-1})$: we matched up pairs of adjacent intervals that differed by the amounts of volume we proposed to add, and compared their price outcomes. We then grouped these pairs of observations according to the volume traded in the previous interval, to see how the difference in price outcomes for the fixed volume shifts behaved as a function of that feature.

The plots here give a representative sample of the types of plots we saw, using January 2019 data.

We would have hoped to see a firm relationship here between the x and y axes, like y values mostly decreasing as x values increased. That is clearly not what we're seeing here, which mostly seems to be flat or noise, and is not consistent between the training and the testing data sets.

We tried several similar tests for other feature possibilities, with similarly unencouraging results. This and many others of these tests were performed when our data normalization procedures were a little less developed, so we will likely revisit them in the future as our abilities to combat noise improve. For now, this failed line of research prompted us to define other possible features more simply/coarsely to make them more robust and try to avoid their effects being drowned out by noise. In particular, our time of day feature which only has three possible values is a blunter way of trying to capture the same kind of phenomenon, as overall trading volumes heavily correlate with time of day in this fashion. [Final aside: If you've made it this far into the paper, my condolences. Now please go outside and get some sunshine.]