L06: Algorithmic Trading & Optimal Execution

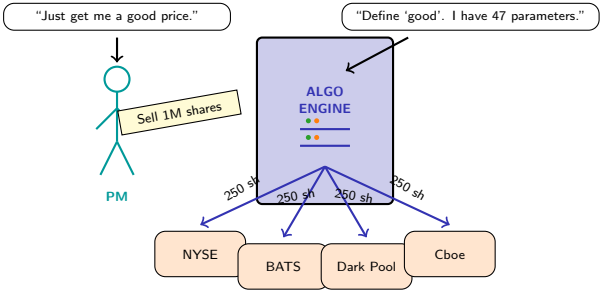
Extended Slides – BSc Digital Finance Course

Digital Finance

What Will You Be Able to Do After This Lecture?

1. Derive the Almgren-Chriss optimal execution trajectory and compute the cost-risk tradeoff for a given order size, volatility, and market impact parameters
2. Implement TWAP and VWAP scheduling engines in Python and measure their execution quality against benchmarks
3. Quantify HFT market making profits using the Avellaneda-Stoikov inventory model and compute optimal bid-ask spreads
4. Decompose transaction costs into spread, market impact, and opportunity components using implementation shortfall attribution
5. Detect backtest overfitting using deflated Sharpe ratios and evaluate MiFID II algorithmic trading requirements

Five objectives: optimal execution theory (1), execution algorithms (2), HFT and market making (3), TCA (4), and backtesting and regulation (5). Mathematical derivations with working Python code and 12 data visualizations.



The PM wanted simplicity. The math had other ideas.

Why Does Slicing a Large Order into Pieces Save Money?

TWAP – For an order of X shares over horizon $[0, T]$ with N intervals:

$$x_k^{\text{TWAP}} = \frac{X}{N}, \quad k = 1, \ldots, N$$

VWAP – Weight each slice by predicted volume profile v_k:

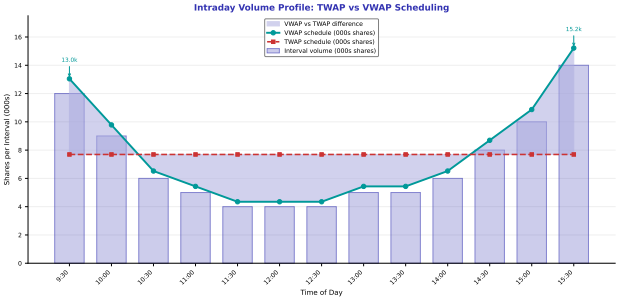
$$x_k^{\text{VWAP}} = X \cdot \frac{v_k}{\sum_{j=1}^{N} v_j}$$

Worked example: $X = 100{,}000$ shares, $N = 5$ intervals, volume profile $v = [0.25, 0.15, 0.10, 0.20, 0.30]$.

- TWAP: each slice $= 100{,}000/5 = 20{,}000$ shares
- VWAP: slices $= [25{,}000,\ 15{,}000,\ 10{,}000,\ 20{,}000,\ 30{,}000]$
- Verification: $25{,}000 + 15{,}000 + 10{,}000 + 20{,}000 + 30{,}000 = 100{,}000$ ✓
- **Key insight:** VWAP front-loads in high-volume periods (less impact per share) and reduces exposure in low-volume periods (where impact is costlier).

TWAP distributes equally across time; VWAP distributes proportionally to volume. VWAP dominates when volume is predictable, TWAP when it is not.

How Does Intraday Volume Shape the Optimal Execution Schedule?



Illustrative 100k-share execution. Volume pattern based on typical U-shaped intraday profile.

- Intraday volume follows a well-known **U-shape**: high at open, low midday, high at close
- VWAP schedule (teal) follows the volume curve – trades more when the market is deepest
- TWAP schedule (red dashed) is flat – ignores volume information entirely
- The shaded area between the two curves shows the **scheduling difference**
- VWAP reduces per-share impact cost by concentrating execution in liquid periods

Intraday volume follows a well-known U-shape. VWAP exploits this by trading more when the market is deepest, reducing per-share impact cost.

What Is the True Cost of Executing a Large Order?

Almgren-Chriss cost functional for executing X shares over $[0, T]$ with trading rate $\dot{x}(t)$ and remaining inventory $x(t)$:

$$J[x] = \underbrace{\gamma \int_0^T \dot{x}(t)\, x(t)\, dt}_{\text{permanent impact cost}} + \underbrace{\eta \int_0^T |\dot{x}(t)|^2\, dt}_{\text{temporary impact cost}}$$

Mean-variance objective: Minimize expected cost plus risk aversion λ times variance:

$$\min_{x(\cdot)} \; E[J] + \lambda \cdot \mathrm{Var}[J] = \gamma X^2/2 + \eta \int_0^T \dot{x}^2\, dt + \lambda\sigma^2 \int_0^T x(t)^2\, dt$$

- **Permanent impact:** $g(v) = \gamma|v|$ (linear in trade rate). Each share traded permanently shifts the price.
- **Temporary impact:** $h(v) = \eta|v|$ (linear, transient). Reflects temporary liquidity displacement.
- **Parameters:** γ = permanent impact coefficient, η = temporary impact coefficient, σ = asset volatility, λ = risk aversion.

The Almgren-Chriss model decomposes execution cost into permanent impact (information) and temporary impact (liquidity). The trader's problem is to minimize cost plus risk.

Can You Compute the Optimal Execution Trajectory in 20 Lines?

```python
import numpy as np
def almgren_chriss(X, T, N, sigma, gamma,
                   eta, lam):
    """Optimal execution trajectory."""
    dt = T / N
    kappa = np.sqrt(lam * sigma**2 / eta)
    # Optimal remaining inventory
    t = np.linspace(0, T, N + 1)
    x = X * np.sinh(kappa * (T - t)) / \
        np.sinh(kappa * T)
    # Trading rate per interval
    trades = -np.diff(x)
    # Expected cost components
    perm = gamma * np.sum(trades * x[:-1])
    temp = eta * np.sum(trades**2) / dt
    risk = lam * sigma**2 * dt * \
           np.sum(x[:-1]**2)
    return x, trades, perm, temp, risk
x, tr, p, t_, r = almgren_chriss(
    X=1e5, T=1.0, N=20, sigma=0.02,
    gamma=2.5e-7, eta=2.5e-6, lam=1e-6)
print(f"Perm: ${p:.0f} Temp: ${t_:.0f}")
print(f"Risk: ${r:.0f} Total: ${p+t_+r:.0f}")
```

- The **sinh trajectory** is the hallmark of Almgren-Chriss
- High risk aversion (λ large) $\Rightarrow$ large κ $\Rightarrow$ aggressive front-loading
- Zero risk aversion ($\lambda \to 0$) $\Rightarrow$ $\kappa \to 0$ $\Rightarrow$ TWAP
- kappa controls the curvature of the optimal path
- Each cost component is computed separately for attribution
- The function returns the full trajectory, trade sizes, and cost breakdown

The sinh(kappa*(T-t)) shape is the hallmark of Almgren-Chriss. High risk aversion curves the trajectory toward aggressive early execution; zero risk aversion yields TWAP.

What Shape Does the Optimal Execution Path Take?

Euler-Lagrange solution to the Almgren-Chriss mean-variance problem:

$$x^*(t) = X \cdot \frac{\sinh\big(\kappa(T-t)\big)}{\sinh(\kappa T)}, \quad \kappa = \sqrt{\frac{\lambda\sigma^2}{\eta}}$$

Optimal trading rate:

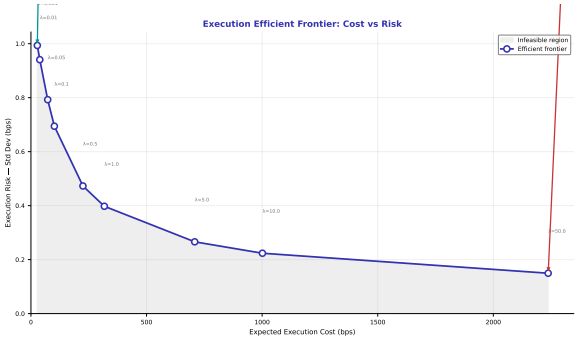
$$\dot{x}^*(t) = -X\kappa \cdot \frac{\cosh\big(\kappa(T-t)\big)}{\sinh(\kappa T)}$$

Worked example: $X = 100{,}000$, $T = 1$ day, $\sigma = 0.02$, $\eta = 2.5 \times 10^{-6}$, $\lambda = 10^{-6}$.

- $\kappa = \sqrt{(10^{-6})(0.02)^2/(2.5 \times 10^{-6})} = \sqrt{(4 \times 10^{-10})/(2.5 \times 10^{-6})} = \sqrt{1.6 \times 10^{-4}} = 0.01265$
- At $t = 0$: $x^*(0) = 100{,}000 \cdot \sinh(0.01265)/\sinh(0.01265) = 100{,}000$ ✓
- At $t = T/2$: $x^*(0.5) = 100{,}000 \cdot \sinh(0.006325)/\sinh(0.01265) \approx 50{,}000$ (nearly linear – low κ)
- **Limiting cases:** $\lambda \to 0 \Rightarrow \kappa \to 0 \Rightarrow x^*(t) = X(1 - t/T)$ (TWAP). $\lambda \to \infty \Rightarrow$ immediate execution.

The parameter kappa controls trajectory curvature. Small kappa (low risk aversion) yields near-TWAP. Large kappa (high risk aversion) front-loads execution aggressively.

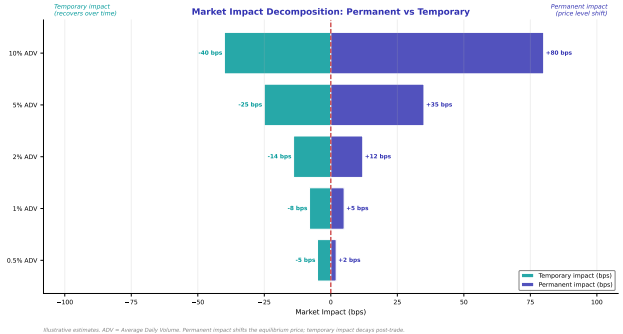
Where Is the Sweet Spot Between Execution Cost and Execution Risk?



- The **execution efficient frontier** plots expected cost vs. execution risk for different λ
- TWAP ($\lambda \to 0$): low cost, high risk – exposed to adverse price moves
- Immediate execution ($\lambda \to \infty$): high cost, zero risk – maximum market impact
- The frontier is **convex** – analogous to the Markowitz efficient frontier

Each execution strategy sits on or above the efficient frontier curve.

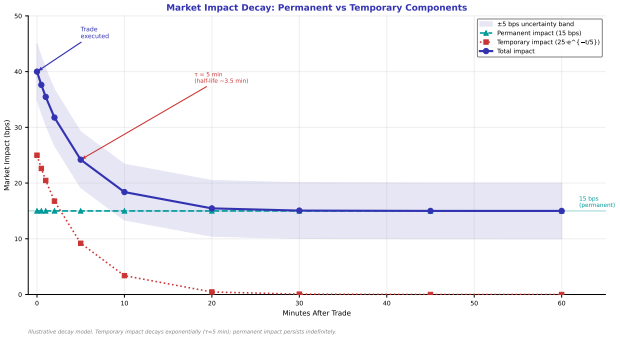
How Much of Your Execution Cost Is Permanent vs Temporary?



- For small orders ($< 1\%$ ADV), **temporary impact dominates** – the order book refills quickly
- For large orders ($> 5\%$ ADV), **permanent impact** becomes the primary cost – information leakage
- The **square-root law** predicts impact $\sim \sqrt{\text{order size}/\text{ADV}}$
- Permanent impact bars extend right; temporary impact bars extend left from center
- Calibrating these components from real data requires careful event-study methodology

For small orders (¡ 1% ADV), temporary impact dominates. For large orders (¿ 5% ADV), permanent impact becomes the primary cost. The square-root law predicts impact grows as the square root of participation rate.

How Quickly Does Market Impact Fade After a Large Trade?



Illustrative decay model. Temporary impact decays exponentially (τ=5 min); permanent impact persists indefinitely.

- **Temporary impact** decays exponentially as the order book refills after the trade
- **Permanent impact** persists indefinitely – the trade revealed information about value
- Total impact = permanent + temporary at each point in time
- The confidence band reflects uncertainty in decay rate estimation
- Decay half-life varies: seconds for liquid large-caps, hours for small-caps
- The decay rate is a key calibration parameter for impact models

Temporary impact decays as the order book replenishes. Permanent impact persists because the trade revealed information about the asset's value. The decay rate is a key calibration parameter.

Can You Build a VWAP Engine That Adapts to Live Volume?

```python
import numpy as np
def vwap_engine(total_qty, vol_profile,
                actual_vol, n_intervals):
    """Adaptive VWAP with live tracking."""
    target = total_qty * vol_profile / \
             vol_profile.sum()
    filled = np.zeros(n_intervals)
    cumul_target = np.cumsum(target)
    cumul_actual_vol = np.cumsum(actual_vol)
    cumul_filled = 0
    for i in range(n_intervals):
        # How far behind/ahead are we?
        gap = cumul_target[i] - cumul_filled
        # Adjust: trade more if behind
        clip = min(gap * 1.2, actual_vol[i]*0.05)
        filled[i] = max(0, clip)
        cumul_filled += filled[i]
    participation = filled / (actual_vol + 1)
    vwap_slip = np.abs(
        cumul_filled - cumul_target[-1])
    return filled, participation, vwap_slip
vol = np.array([8,5,3,3,4,6,5,4,5,7]) * 1e4
pred = np.array([7,5,3,4,4,5,5,4,5,8]) * 1e4
f, p, s = vwap_engine(50000, pred, vol, 10)
print(f"Slippage: {s:.0f} shares")
```

- **Adaptive VWAP** adjusts to real-time volume deviations from the predicted profile
- The engine tracks cumulative target vs. cumulative fill at each interval
- If behind schedule, it increases the fill rate (up to $1.2\times$ the gap)
- **Participation cap** of 5% prevents the algorithm from dominating any interval
- VWAP is the most popular execution benchmark – used by $>60\%$ of institutional orders
- Production systems add order-type selection and venue routing

Adaptive VWAP tracks actual volume in real time and adjusts the schedule to stay on target. The 5% participation cap prevents the algorithm from dominating any single interval.

How Does a Market Maker Set the Optimal Bid-Ask Spread?

Avellaneda-Stoikov reservation price: A market maker with inventory q and risk aversion γ sets:

$$r(s, q, t) = s - q\gamma\sigma^2(T - t)$$

where s = mid-price, σ = volatility, T = end of trading day.
Optimal spread:

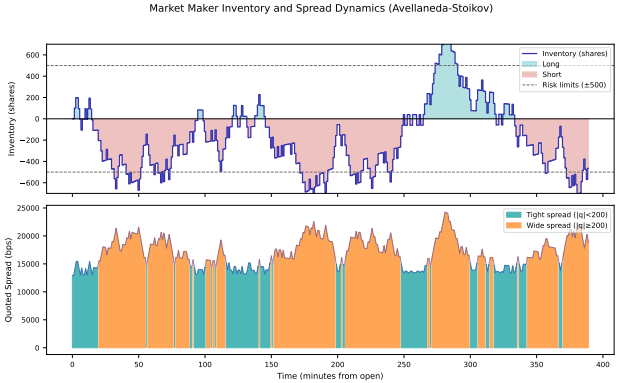
$$\delta^*(q, t) = \gamma\sigma^2(T - t) + \frac{2}{\gamma}\ln\Big(1 + \frac{\gamma}{k}\Big)$$

where k = order arrival rate parameter. **Optimal quotes:** bid $= r - \delta^*/2$, ask $= r + \delta^*/2$.
Worked example ($\gamma = 0.1$): $s = \$50.00$, $q = +200$ (long), $\sigma = 0.003/\text{min}$, $T - t = 60$ min, $k = 1.5$.

- Reservation: $r = 50.00 - 200 \times 0.1 \times 0.003^2 \times 60 = 50.00 - 0.0108 = 49.989$
- Spread: $\delta^* = 0.1 \times 0.000009 \times 60 + (2/0.1) \times \ln(1 + 0.1/1.5) = 0.000054 + 20 \times 0.06454 = 1.291$
- Quotes: bid $= 49.989 - 0.645 = 49.34$, ask $= 49.989 + 0.645 = 50.63$
- **Interpretation:** Positive inventory shifts reservation below mid (maker wants to sell). Spread widens with volatility and risk aversion.

Avellaneda-Stoikov shows that optimal spread depends on inventory, volatility, time remaining, and order arrival intensity. The reservation price shifts away from mid to shed inventory risk.

How Does a Market Maker's Inventory Drive Their Quotes?



- **Top panel:** Inventory oscillates around zero with occasional buildups
- When inventory is extreme (far from zero), the market maker faces **adverse selection risk**
- **Bottom panel:** Spread widens when inventory is risky, tightens near zero
- Teal shading = long inventory; red shading = short inventory
- The Avellaneda-Stoikov model captures this behavior through the $q\gamma\sigma^2(T-t)$ term
- Horizontal dashed lines show risk limits beyond which the maker should hedge

Market makers widen spreads when their inventory is risky and tighten when it is balanced. The Avellaneda-Stoikov model formalizes this intuition as optimal stochastic control.

How Much Money Does a Microsecond of Speed Advantage Generate?

Latency arbitrage setup: A fast trader observes a price change on exchange A before it propagates to exchange B. The stale quote on B can be picked off.

Expected PnL per race:

$$E[\text{PnL}] = P(\text{win}) \cdot \underbrace{|\Delta p|}_{\text{stale quote}} \cdot Q - \underbrace{c}_{\text{cost/attempt}}$$

Annualized revenue:

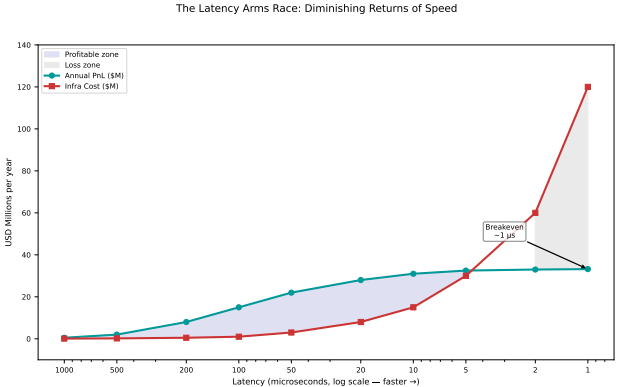
$$\text{Rev} = N_{\text{races/day}} \times P(\text{win}) \times E[|\Delta p|] \times \bar{Q} \times 252$$

Worked example: 50,000 races/day, $P(\text{win}) = 0.60$, $E[|\Delta p|] = \$0.005$, $\bar{Q} = 500$ shares, cost/race $= \$0.50$.

- PnL per race: $0.60 \times 0.005 \times 500 - 0.50 = 1.50 - 0.50 = \$1.00$
- Daily: $50{,}000 \times 1.00 = \$50{,}000$
- Annual: $\$50{,}000 \times 252 = \$12.6\text{M}$
- Verification: Revenue $= 50{,}000 \times 0.60 \times 0.005 \times 500 = \$75{,}000$. Cost $= 50{,}000 \times 0.50 = \$25{,}000$. Net $= \$50{,}000$/day ✓
- **Key insight:** If $P(\text{win})$ drops from 0.60 to 0.40, revenue drops by 1/3 and the desk becomes unprofitable after fixed costs.

Latency arbitrage revenue is linear in the number of races won. A 20 percentage point drop in win rate can turn a profitable desk into a loss-maker.

At What Point Does Faster Hardware Stop Paying for Itself?



- **Teal line:** Annual PnL shows steep gains from 1000μs to 100μs latency
- **Red line:** Infrastructure cost rises steeply as latency decreases below 10μs
- Shaded purple region = profit zone; gray region = loss zone
- The **breakeven point** marks where marginal cost exceeds marginal revenue
- Going from 1ms to 100μs is worth millions; 10μs to 1μs may not cover hardware
- Budish et al. argue this arms race is socially wasteful – frequent batch auctions would eliminate it

The latency-profit curve exhibits severe diminishing returns. Going from 1ms to 100us is worth millions; going from 10us to 1us may not cover the hardware cost.

Can You Simulate a Market Maker's Day in 20 Lines?

```python
import numpy as np
def simulate_mm(T=390, sigma=0.003,
                gamma=0.1, k=1.5, q0=0):
    """Avellaneda-Stoikov market maker."""
    s = 50.0 # mid-price
    q, pnl = q0, 0.0
    inv_path, spread_path = [], []
    for t in range(T):
        tau = T - t
        r = s - q * gamma * sigma**2 * tau
        delta = (gamma * sigma**2 * tau
                 + 2/gamma * np.log(1+gamma/k))
        bid, ask = r - delta/2, r + delta/2
        # Random arrivals
        if np.random.rand() < 0.3: # cust buy
            q -= 100; pnl += ask * 100
        if np.random.rand() < 0.3: # cust sell
            q += 100; pnl -= bid * 100
        s += sigma * np.random.randn()
        inv_path.append(q)
        spread_path.append(delta)
    pnl += q * s # mark-to-market
    return pnl, inv_path, spread_path
pnl, inv, sp = simulate_mm()
print(f"PnL: ${pnl:,.0f} Final q: {inv[-1]}")
```

- When a customer **buys** (hits the ask), the maker **sells**: q decreases, PnL increases by ask$\times$qty
- When a customer **sells** (hits the bid), the maker **buys**: q increases, PnL decreases by bid$\times$qty
- The spread δ is inventory-dependent via the Avellaneda-Stoikov formula
- Higher γ widens spreads (more risk averse)
- Higher k tightens spreads (more order flow expected)
- Final mark-to-market converts remaining inventory to PnL at current mid-price

This simplified simulator captures the core Avellaneda-Stoikov dynamics: inventory-dependent quoting, volatility-scaled spreads, and stochastic order arrivals.

Where Did the Money Go Between Decision and Execution?

Implementation shortfall (Perold, 1988): Total cost = Paper portfolio return − Actual portfolio return.
Three-component decomposition:

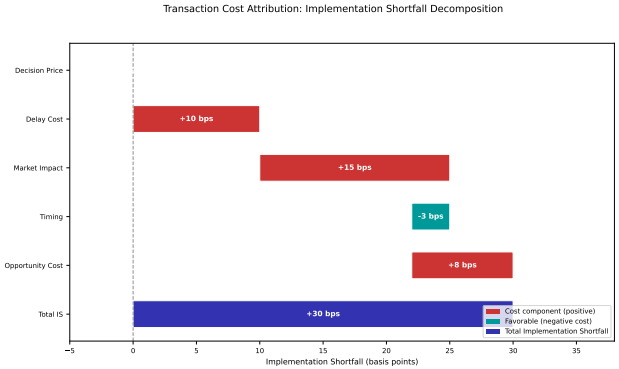
$$\text{IS} = \underbrace{(P_{\text{arrival}} - P_{\text{decision}}) \cdot X}_{\text{delay cost}} + \underbrace{(\bar{P}_{\text{exec}} - P_{\text{arrival}}) \cdot X_{\text{filled}}}_{\text{market impact}} + \underbrace{(P_{\text{close}} - P_{\text{arrival}}) \cdot X_{\text{unfilled}}}_{\text{opportunity cost}}$$

Worked example: Decision to buy 10,000 shares at $P_{\text{decision}} = \$50.00$.

- $P_{\text{arrival}} = \$50.10$ (market moved while order was queued)
- $\bar{P}_{\text{exec}} = \$50.25$ (impact during execution), $P_{\text{close}} = \$50.40$
- Filled: 8,000 shares; Unfilled: 2,000 shares
- Delay cost: $(50.10 - 50.00) \times 10{,}000 = \$1{,}000$
- Market impact: $(50.25 - 50.10) \times 8{,}000 = \$1{,}200$
- Opportunity cost: $(50.40 - 50.10) \times 2{,}000 = \$600$
- **Total IS:** $\$1{,}000 + \$1{,}200 + \$600 = \$2{,}800 = 56$ bps (vs. $10{,}000 \times \$50.00$)
- Verification: Paper PnL $= (50.40 - 50.00) \times 10{,}000 = \$4{,}000$. Actual PnL $= (50.40 - 50.25) \times 8{,}000 = \$1{,}200$. IS $= 4{,}000 - 1{,}200 = \$2{,}800$ ✓

Implementation shortfall decomposes total execution cost into delay (order queue time), impact (price movement caused by your trading), and opportunity (unfilled quantity that would have been profitable).

Which Component of Transaction Cost Eats Most of Your Alpha?



- The waterfall starts at the **decision price** and accumulates cost components
- **Market impact** ($+15$ bps) is the largest single component – and the only one an algorithm can directly minimize
- **Delay cost** ($+10$ bps) reflects the time between decision and first execution
- **Timing** (-3 bps) can be favorable when the market moves in your direction
- **Opportunity cost** ($+8$ bps) captures the cost of unfilled quantity
- Total IS $= 30$ bps in this example

TCA attribution decomposes execution shortfall into actionable components. Market impact is the only component an algorithm can directly minimize.

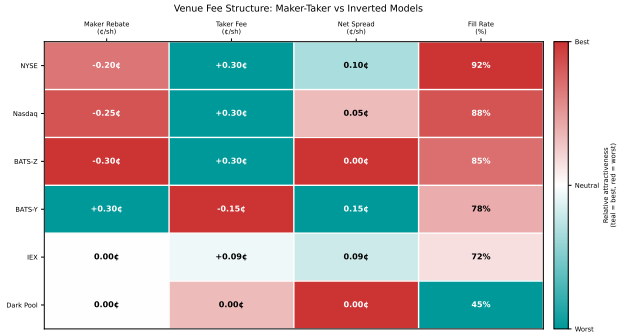
Can You Build an Implementation Shortfall Calculator?

```python
import numpy as np
def impl_shortfall(p_dec, p_arr, p_exec,
                   p_close, qty_ord, qty_fill):
    """Perold (1988) IS decomposition."""
    unfilled = qty_ord - qty_fill
    delay = (p_arr - p_dec) * qty_ord
    impact = (np.mean(p_exec) - p_arr) \
             * qty_fill
    opp = (p_close - p_arr) * unfilled
    total = delay + impact + opp
    bps = total/(p_dec * qty_ord) * 1e4
    return {"delay_$": delay,
            "impact_$": impact,
            "opp_$": opp, "total_$": total,
            "total_bps": bps}
result = impl_shortfall(
    p_dec=50.00, p_arr=50.10,
    p_exec=[50.15, 50.20, 50.30, 50.35],
    p_close=50.40,
    qty_ord=10000, qty_fill=8000)
for k, v in result.items():
    print(f"{k:>12}: {v:>10.1f}")
```

- Inputs: decision price, arrival price, execution prices (per child order), close
- The function computes each component separately for attribution
- Aggregates across child orders via `np.mean(p_exec)`
- Returns both dollar cost and basis points
- Under **MiFID II RTS 28**, brokers must report execution quality data quarterly
- This computation is now routine at every institutional trading desk

Implementation shortfall is the industry-standard TCA metric. MiFID II requires brokers to report execution quality data quarterly, making this computation routine.

Why Does Your Broker Route Orders to One Exchange Instead of Another?



- Each venue charges different **maker** (provide liquidity) and **taker** (remove liquidity) fees
- **Maker-taker model:** rebate for adding liquidity, fee for removing it (NYSE, Nasdaq)
- **Inverted venues** (BATS-Y): pay takers, charge makers – attracts aggressive flow
- **IEX** uses a 350μs speed bump to protect displayed quotes
- **Dark pools:** no display obligation but potential information leakage risk
- The cheapest venue is not always the best venue

Smart order routing must balance visible cost (fees) against hidden cost (adverse selection, information leakage). The cheapest venue is not always the best venue.

How Do You Mathematically Optimize Where to Send Each Child Order?

SOR objective: Minimize total execution cost across M venues:

$$\min_{q_1,\ldots,q_M} \sum_{m=1}^{M} [f_m(q_m) + g_m(q_m) + c_m \cdot q_m]$$

subject to $\sum_{m=1}^{M} q_m = Q$ (total quantity), $0 \le q_m \le D_m$ (depth at venue m).
Where: $f_m(q_m)$ = market impact at venue m, $g_m(q_m)$ = adverse selection cost, c_m = fee per share, D_m = visible depth.
Worked example: $Q = 5{,}000$ shares across 3 venues.

- NYSE: $c_1 = -0.002$ (rebate), $D_1 = 3{,}000$, $f_1(q) = 0.001q^{0.5}$
- BATS: $c_2 = -0.003$ (bigger rebate), $D_2 = 2{,}000$, $f_2(q) = 0.0015q^{0.5}$
- Dark: $c_3 = 0$ (no fee), $D_3 = \infty$, $f_3(q) = 0.0005q^{0.5}$, $g_3(q) = 0.003q$ (adverse selection)
- **Optimal:** Fill $D_2 = 2{,}000$ at BATS (cheapest fee), then $D_1 = 3{,}000$ at NYSE (depth-limited), 0 to dark (adverse selection exceeds benefit).
- Total cost: BATS: $0.0015\sqrt{2000} - 0.003 \times 2000 = 0.067 - 6.00 = -5.93$. NYSE: $0.001\sqrt{3000} - 0.002 \times 3000 = 0.055 - 6.00 = -5.95$. Net $= -11.88$ (rebate revenue).

Smart order routing is a constrained optimization: minimize impact + fees + adverse selection across venues, subject to depth and quantity constraints.

Can You Build a Simple Smart Order Router?

```python
import numpy as np
def smart_router(qty, venues):
    """Greedy SOR: fill cheapest first."""
    alloc = {}
    remaining = qty
    # Sort by total cost per share
    ranked = sorted(venues.items(),
        key=lambda v: v[1]["fee"]
                    + v[1]["impact_per_sh"])
    for name, info in ranked:
        depth = info["depth"]
        fill = min(remaining, depth)
        alloc[name] = fill
        remaining -= fill
        if remaining <= 0:
            break
    return alloc
venues = {
    "NYSE": {"fee":-0.002, "depth":3000,
             "impact_per_sh": 0.005},
    "BATS": {"fee":-0.003, "depth":2000,
             "impact_per_sh": 0.006},
    "Dark": {"fee": 0.000, "depth":9999,
             "impact_per_sh": 0.010}}
print(smart_router(5000, venues))
```

- **Greedy SOR:** fill cheapest venue first, respecting depth limits
- Cost per share = fee + estimated impact per share
- BATS is cheapest ($-0.003 + 0.006 = 0.003$), then NYSE ($-0.002 + 0.005 = 0.003$), then Dark ($0.000 + 0.010 = 0.010$)
- Limitation: ignores adverse selection correlation, queue position, latency
- Production SOR uses **reinforcement learning** to adapt routing in real time
- Best execution regulations require documenting routing decisions

This greedy router captures the core idea: fill cheapest venues first, respecting depth limits. Production SOR adds latency modeling, queue priority, and reinforcement learning.

How Many Backtests Does It Take to Find a Strategy That Only Looks Good?

Multiple testing problem: If you test N strategies, the probability that at least one has SR $>$ threshold by chance:

$$P\left(\max_{i=1}^{N} \widehat{SR}_i > \text{threshold}\right) = 1 - \Phi\left(\frac{\text{threshold} - \mu_{SR}}{\sigma_{SR}}\right)^N$$

Deflated Sharpe Ratio (Bailey & Lopez de Prado, 2014):

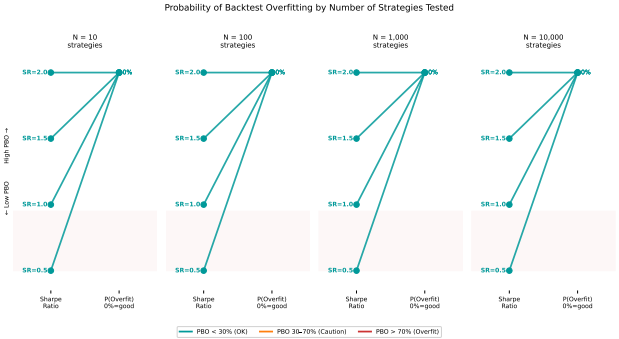
$$DSR = \Phi\left[\frac{(\widehat{SR} - SR_0)\sqrt{T}}{\sqrt{1 - \hat{\gamma}_3 \widehat{SR} + \frac{\hat{\gamma}_4 - 1}{4}\widehat{SR}^2}}\right], \quad SR_0 = \sqrt{\tfrac{2}{T}} \cdot \text{erf}^{-1}\left(1 - \tfrac{1}{N}\right)$$

Worked example: $\widehat{SR} = 1.5$, $T = 252$ days, $N = 100$ strategies, skewness $= -0.5$, kurtosis $= 4.0$.

- $SR_0 = \sqrt{2/252} \times \text{erf}^{-1}(0.99) = 0.0891 \times 1.821 = 0.162$
- Numerator: $(1.5 - 0.162) \times \sqrt{252} = 1.338 \times 15.875 = 21.24$
- Denominator: $\sqrt{1 + 0.75 + 1.6875} = \sqrt{3.4375} = 1.854$
- $DSR = \Phi(21.24/1.854) = \Phi(11.46) \approx 1.00$ – almost certainly a genuine edge
- A strategy with $\widehat{SR} = 0.3$ among $N = 1{,}000$ trials would give $DSR \approx 0.0$ (likely overfitting)

The Deflated Sharpe Ratio adjusts for multiple testing. The more strategies you try, the higher the bar for statistical significance. Most hedge fund backtests fail this test.

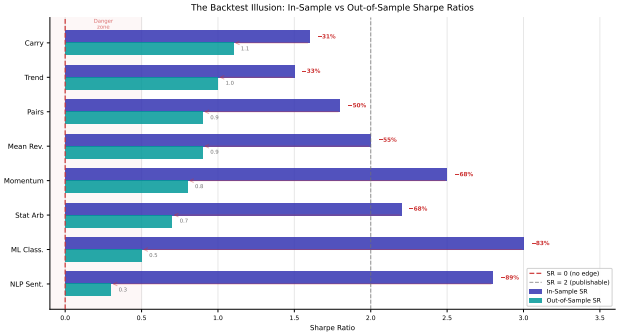
How Do You Tell a Genuine Edge from a Backtest Mirage?



- The slope chart shows how **overfitting probability** changes with the number of strategies tested (N)
- At $N = 10$, even a modest SR $= 0.5$ has low probability of being overfitted
- At $N = 10{,}000$, even SR $= 1.5$ is likely a false discovery
- Teal lines = low overfitting risk; red lines = high risk
- The **combinatorial PBO** method formalizes this intuition
- Hedge funds must track how many strategies they have tested to compute honest DSR

With 1,000 backtests, a Sharpe ratio of 1.5 is more likely a false discovery than a genuine edge. The cure: pre-register hypotheses and use out-of-sample validation.

Why Do Brilliant Backtests Fail in Live Trading?



- Purple bars = **in-sample** Sharpe ratio (backtest); teal bars = **out-of-sample** (live)
- Every strategy shows significant **IS-to-OOS degradation** (50–80% drop typical)
- ML classifiers and NLP sentiment degrade most – more free parameters = more overfitting
- Carry and trend following are most robust – fewer parameters, stronger economic rationale
- The dashed red line at SR = 0 marks the breakeven threshold
- **Walk-forward validation** mitigates but does not eliminate this gap

In-sample performance consistently overstates out-of-sample returns. The degradation is worst for strategies with more free parameters (ML, NLP) and least for simple strategies (carry, trend).

Can You Compute the Probability Your Backtest Is Overfitted?

```python
import numpy as np
from scipy.stats import norm
from scipy.special import erfinv
def deflated_sharpe(sr_hat, T, N,
                    skew=0, kurt=3):
    """Bailey & Lopez de Prado (2014)."""
    sr0 = np.sqrt(2/T) * erfinv(1 - 1/N)
    numer = (sr_hat - sr0) * np.sqrt(T)
    denom = np.sqrt(
        1 - skew * sr_hat
        + (kurt - 1)/4 * sr_hat**2)
    return norm.cdf(numer / denom)
# Test: SR=1.5, 252 days, 100 strategies
dsr = deflated_sharpe(1.5, 252, 100,
                      skew=-0.5, kurt=4)
print(f"DSR (SR=1.5, N=100): {dsr:.4f}")
# What if N=10000 strategies?
dsr2 = deflated_sharpe(1.5, 252, 10000,
                       skew=-0.5, kurt=4)
print(f"DSR (SR=1.5, N=10000): {dsr2:.4f}")
# Marginal case
dsr3 = deflated_sharpe(0.5, 252, 100)
print(f"DSR (SR=0.5, N=100): {dsr3:.4f}")
```

- DSR near 1.0 = likely a **genuine edge** (statistically significant after multiple-testing correction)
- DSR near 0.0 = likely **overfitting** (the Sharpe ratio is explained by the number of trials)
- The function adjusts for non-normal returns via skewness and kurtosis terms
- Tracking N (number of backtests tried) is **essential** – without it, the Sharpe ratio is meaningless
- Connection to the replication crisis in academic finance research

The Deflated Sharpe Ratio is the quantitative antidote to backtest overfitting. If you cannot report how many strategies you tried, your Sharpe ratio is meaningless.

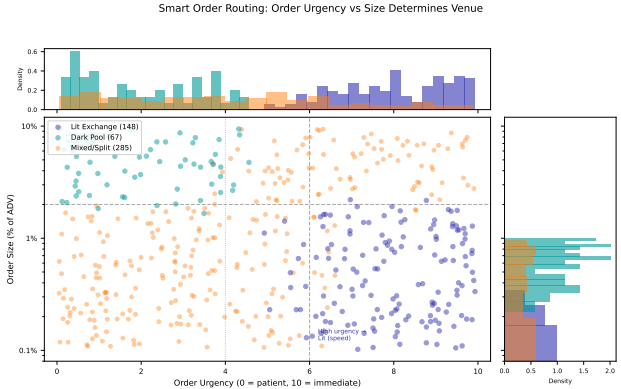
What Does MiFID II Require from Every Algorithm That Trades in Europe?

Requirement	What It Means	Why It Matters
Algo registration (Art. 17)	Register all algo strategies with national regulator	Accountability: regulators know who runs what
Kill switch	Cancel all algo orders instantly	Flash crash prevention: human override
Pre-trade risk controls	Max order size, price collars, throttling	Prevent rogue algos from outsized positions
Post-trade monitoring	Real-time surveillance of algo behavior	Detect manipulation (spoofing, layering)
Business continuity	Tested disaster recovery for algo systems	System failure cannot cascade to markets
Maker obligations (DEA)	Market makers must provide liquidity continuously	Prevent "liquidity mirages" in stress

- **US comparison:** SEC Rule 15c3-5 (market access), CAT (Consolidated Audit Trail)
- **Swiss:** FINMA algo trading circular with similar pre-trade controls
- **Gap:** ML/AI-driven trading not yet fully covered by any jurisdiction

MiFID II is the most comprehensive algorithmic trading regulation globally. It requires algo registration, kill switches, pre-trade controls, and continuous market-making obligations.

How Does a Smart Order Router Choose Between Lit Exchanges and Dark Pools?



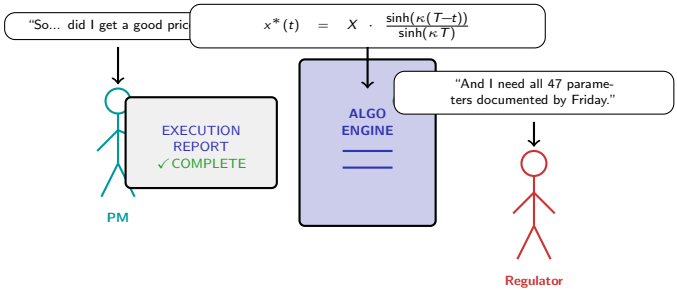
- The hexbin plot maps **order urgency** (x) vs. **order size** (y) colored by dominant venue
- **Lit exchanges** (purple): high urgency + small size – speed matters, impact is low
- **Dark pools** (teal): low urgency + large size – minimize information leakage
- **Mixed/internalized** (orange): medium region – broker discretion
- Large passive orders prefer dark pools because displayed quotes invite front-running
- Best-execution regulations require documenting the routing rationale

Smart order routing is a multi-objective optimization: minimize cost, maximize fill rate, minimize information leakage, and satisfy regulatory best-execution obligations.

How Has Regulation Evolved to Keep Pace with Algorithmic Trading?

- US events (purple) and EU events (teal) on two horizontal strip lanes
- Crisis events (red diamonds) span both lanes – they triggered regulatory responses
- **Pattern:** Regulation always follows crisis, never precedes it
- Flash Crash (2010) → circuit breakers; Knight Capital (2012) → pre-trade risk controls
- **MiFID II** (2018) is the most comprehensive algo trading framework globally
- **Gap:** ML/AI trading strategies not yet fully covered by any regime

Algorithmic trading regulation follows a crisis-response pattern: Flash Crash (2010) led to circuit breakers, Knight Capital (2012) led to pre-trade risk controls, and MiFID II (2018) created a comprehensive framework.



The PM wanted a price. The algo gave a trajectory. The regulator wanted both, in writing.

References and Further Reading

1. Almgren, R. & Chriss, N. (2001). Optimal execution of portfolio transactions. *Journal of Risk*, 3(2), 5–39.
2. Avellaneda, M. & Stoikov, S. (2008). High-frequency trading in a limit order book. *Quantitative Finance*, 8(3), 217–224.
3. Perold, A. (1988). The implementation shortfall: Paper versus reality. *Journal of Portfolio Management*, 14(3), 4–9.
4. Bailey, D. & Lopez de Prado, M. (2014). The Deflated Sharpe Ratio. *Journal of Portfolio Management*, 40(5), 94–107.
5. Cont, R., Kukanov, A. & Stoikov, S. (2014). The price impact of order book events. *Journal of Financial Econometrics*, 12(1), 47–88.
6. MiFID II/MiFIR technical standards: RTS 6 (algorithmic trading), RTS 28 (execution quality reporting).

Six foundational references: optimal execution (1), market making (2), TCA (3), backtest validity (4), market impact (5), and regulation (6).