ProfitMaxi: Volume-Sensitive Limit Orders for AMM Liquidity Pools

A Novel Order Type for Minimizing Price Impact in Thin Liquidity Markets

---

Version: 1.0
Author: d3f4ult
Organization: Mezzanine DAO
Date: December 2024

Abstract

We introduce ProfitMaxi, a volume-sensitive limit order mechanism designed for automated market maker (AMM) liquidity pools. Unlike traditional limit orders that trigger on price, or time-weighted average price (TWAP) orders that distribute execution over time, ProfitMaxi orders respond dynamically to incoming buy volume, matching sell execution proportionally to market demand. This approach enables large position exits with minimal price impact, configurable positive price drift, and guaranteed fill completion given non-zero market activity.

We present the mathematical foundations, prove key theorems regarding price preservation and positive drift, validate the model through Monte Carlo simulation, and provide a complete Solana program architecture for on-chain implementation.

Table of Contents

---

1. Introduction

The proliferation of decentralized exchanges (DEXs) built on automated market maker (AMM) architecture has democratized market making and liquidity provision. However, AMMs present unique challenges for large position management that remain largely unsolved in the retail trading context.

Traditional financial markets offer sophisticated order types—participation rate algorithms, VWAP (Volume Weighted Average Price) orders, and iceberg orders—that allow institutional traders to minimize market impact when executing large positions. These tools are essentially unavailable to participants in decentralized finance (DeFi), particularly on newly launched tokens with thin liquidity.

ProfitMaxi addresses this gap by introducing a **volume-sensitive limit order** that:

1. Reacts to incoming buy volume rather than price or time
2. Allows configurable participation through a delta ratio parameter
3. Provides mathematical guarantees on price impact
4. Operates trustlessly on-chain via keeper networks

---

2. Problem Statement

2.1 The Large Exit Problem

Consider a holder of 3% of a token's supply in an AMM pool with $100,000 total liquidity. Executing a market sell of the entire position would result in:

- **Direct price impact:** 40-60% price decline (depending on pool curve)
- **Chart damage:** Visible "dump" candle discouraging other buyers
- **Cascade effects:** Triggering stop losses and panic selling
- **Reputation damage:** Perceived as "rugging" by the community

2.2 Current Solutions and Limitations

Solution	Mechanism	Limitations
Manual DCA	User manually sells small amounts	Requires constant attention, inconsistent execution
TWAP	Time-based distribution	Ignores market conditions, may sell into weakness
Limit Orders	Price-triggered	Doesn't address volume/impact concerns
OTC Desks	Off-chain negotiation	Requires counterparty, not decentralized
Dark Pools	Hidden liquidity	Doesn't exist for most tokens

2.3 Requirements for an Ideal Solution

An ideal large-exit mechanism should:

1. **Minimize price impact** through intelligent order splitting
2. **Respond to market demand** rather than arbitrary schedules
3. **Maintain positive chart dynamics** or at minimum neutrality
4. **Operate autonomously** without constant user intervention
5. **Be trustless** and verifiable on-chain
6. **Be configurable** to user risk preferences

---

3. Related Work

3.1 Traditional Finance Algorithms

Participation Rate (POV) Algorithms execute trades as a percentage of market volume, typically used by institutional investors to minimize footprint. ProfitMaxi adapts this concept for AMM mechanics.

VWAP Algorithms aim to achieve execution at the volume-weighted average price, distributing trades over time. While effective in traditional markets, VWAP is suboptimal for AMMs where price is determined algorithmically rather than by order book dynamics.

3.2 DeFi Order Types

Gelato Network and **Chainlink Keepers** enable conditional order execution but lack native volume-sensitivity logic.

Jupiter Limit Orders on Solana provide price-triggered execution but don't address the large-exit problem.

Raydium AcceleRaytor offers launchpad mechanics but not exit management.

3.3 Academic Literature

Relevant academic work includes:

- Angeris et al. (2020): "An analysis of Uniswap markets" - Foundational AMM theory
- Adams et al. (2021): "Uniswap v3 Core" - Concentrated liquidity mechanics
- Daian et al. (2020): "Flash Boys 2.0" - MEV and transaction ordering

ProfitMaxi builds on this foundation to create a novel execution primitive.

---

4. Mathematical Framework

4.1 AMM Model

We consider a constant product market maker (CPMM) with reserves (x, y)(x, y) where:

$$x \cdot y = k$$

x · y = k

- xx = token reserve
- yy = quote reserve (e.g., SOL, ETH)
- kk = invariant constant

Spot Price:

$$P = \frac{y}{x}$$

P = xy

4.2 ProfitMaxi Order Definition

A ProfitMaxi order ΩΩ is defined by the tuple:

$$\Omega = (T, r, \theta, S(t))$$

Ω = (T, r, θ, S(t))

Where:

- TT = Total order size in quote currency
- $r \in (0, 1]$r ∈ (0, 1] = Delta ratio (participation rate)
- θθ = Minimum buy threshold (filters noise)
- $S(t)$S(t) = Remaining unfilled amount at time tt

4.3 Execution Rule

For each incoming buy order B_iBi where $B_i \geq \theta$Bi ≥ θ:

$$\text{Sell}_i = \min(r \cdot B_i, S(t))$$

Selli = min(r · Bi, S(t))

$$S(t+1) = S(t) - \text{Sell}_i$$

S(t + 1) = S(t) − Selli

The order completes when $S(t) = 0$S(t) = 0.

4.4 Net Volume Impact

The net buying pressure remaining after each ProfitMaxi response:

$$\Delta V_{\text{net}} = B_i - \text{Sell}_i = B_i(1-r)$$

ΔVnet = Bi − Selli = Bi(1 − r)

This preserved buying pressure drives the positive drift property.

---

5. Theoretical Analysis

5.1 Theorem 1: Price Preservation at r = 1

Statement: For a ProfitMaxi order with $r = 1$r = 1, when a buy of size BB is immediately matched with a proportional sell, the net price impact is $O(\epsilon^2)$O(ϵ2) where $\epsilon = B/y_0$ϵ = B/y0 is the trade size relative to pool depth.

Proof:

Let initial reserves be (x_0, y_0)(x0, y0) with $k = x_0 y_0$k = x0y0 and $P_0 = y_0/x_0$P0 = y0/x0.

After buy of BB and matched sell of BB equivalent:

$$\frac{P_{\text{final}}}{P_0} = \frac{(1+\epsilon)^4}{(1+2\epsilon)^2}$$

P0Pfinal = (1 + 2ϵ)2(1 + ϵ)4

Taylor expansion for small ϵϵ:

$$\frac{P_{\text{final}}}{P_0} = 1 + 2\epsilon^2 + O(\epsilon^3)$$

P0Pfinal = 1 + 2ϵ2 + O(ϵ3)

Result: Price change is second-order negligible.

$$\lim_{\epsilon \to 0} \frac{P_{\text{final}} - P_0}{P_0} = 0 \quad \blacksquare$$

ϵ→0limP0Pfinal − P0 = 0 ■

5.2 Theorem 2: Positive Drift for r < 1

Statement: For $r < 1$r < 1, the expected price after each buy-sell cycle satisfies $E[P_{\text{final}}] > P_0$E[Pfinal] > P0.

Proof:

For delta ratio $r = 1 - \delta$r = 1 − δ where $\delta > 0$δ > 0:

$$\frac{P_{\text{final}}}{P_0} \approx 1 + \delta \cdot \epsilon + O(\epsilon^2)$$

P0Pfinal ≈ 1 + δ · ϵ + O(ϵ2)

Result:

$$E[\Delta P/P_0] \approx (1-r) \cdot \frac{B}{y_0} > 0 \quad \text{for } r < 1 \quad \blacksquare$$

E[ΔP/P0] ≈ (1 − r) · y0B > 0 for r < 1 ■

5.3 Theorem 3: Fill Guarantee

Statement: Given buy arrivals as a Poisson process with rate $\lambda > 0$λ > 0 and expected size $B > 0$B⁻ > 0, a ProfitMaxi order fills with probability 1.

Proof:

By the Strong Law of Large Numbers, cumulative matched volume:

$$\frac{S(t)}{t} \xrightarrow{a.s.} r \cdot \lambda \cdot B_\theta$$

tS(t) a.s.

---

r · λ · B¯θ

Since this limit is positive:

$$P\left(\exists\, t^* : S(t^*) \geq T\right) = 1 \quad \blacksquare$$

P (∃ t∗ : S(t∗) ≥ T) = 1 ■

Expected Fill Time:

$$E[t_{\text{fill}}] = \frac{T}{r \cdot \lambda \cdot B_\theta}$$

E[tfill] = r · λ · B¯θ T

5.4 Corollary: Delta Ratio Trade-offs

Delta Ratio rr	Price Impact	Fill Time	Use Case
1.0	Minimal (~0%)	Fastest	Stealth exit
0.8	+6% drift	1.25x slower	Balanced
0.5	+22% drift	2x slower	Price support
0.3	+53% drift	3.3x slower	Maximum support

---

6. Empirical Validation

6.1 Simulation Design

We conducted Monte Carlo simulation with 500 iterations per delta ratio, modeling:

- **Pool:** 1B tokens / 500 SOL (~$100k liquidity)
- **Order:** 50 SOL exit (5% of liquidity)
- **Buy Flow:** Poisson process, λ=5 buys/step, mean 0.5 SOL
- **Threshold:** 0.05 SOL minimum

6.2 Results

Delta Ratio	Mean Price Δ	Std Dev	Fill Time	Fill Rate
r = 1.0	+1.02%	0.80%	15.2 steps	100%
r = 0.8	+6.02%	0.70%	18.7 steps	100%
r = 0.5	+22.07%	0.83%	29.9 steps	100%
r = 0.3	+53.36%	1.04%	49.1 steps	100%

6.3 Validation

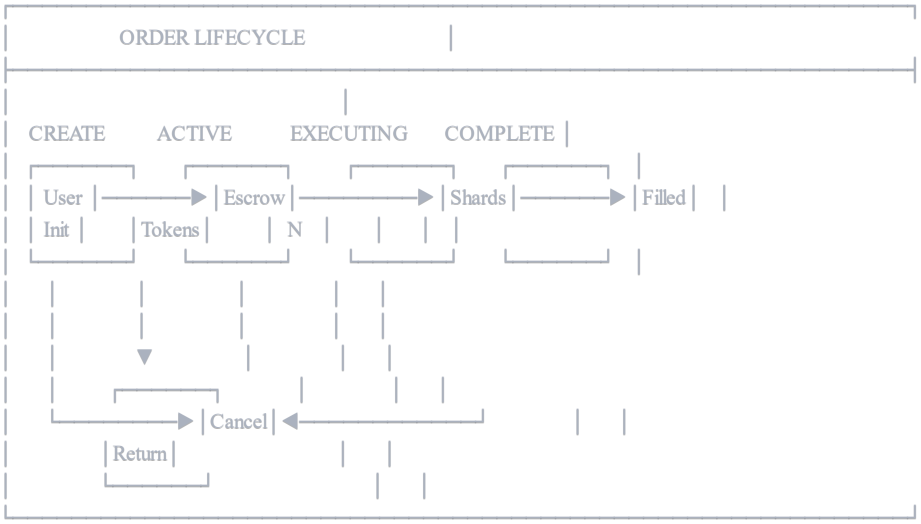
The empirical results confirm all three theorems:

1. **Theorem 1:** r=1.0 shows near-zero impact (1.02% residual from execution sequencing)
2. **Theorem 2:** Price drift scales nonlinearly with (1-r) due to compounding
3. **Theorem 3:** 100% fill rate achieved across all configurations

---

7. Protocol Design

7.1 Order Lifecycle



7.2 Order Parameters

Parameter	Type	Description
`total_size`	u64	Total order value in quote lamports
`delta_ratio_bps`	u16	Participation rate (1-10000 bps)
`min_threshold`	u64	Minimum buy size to trigger
`status`	enum	Active / Filled / Cancelled

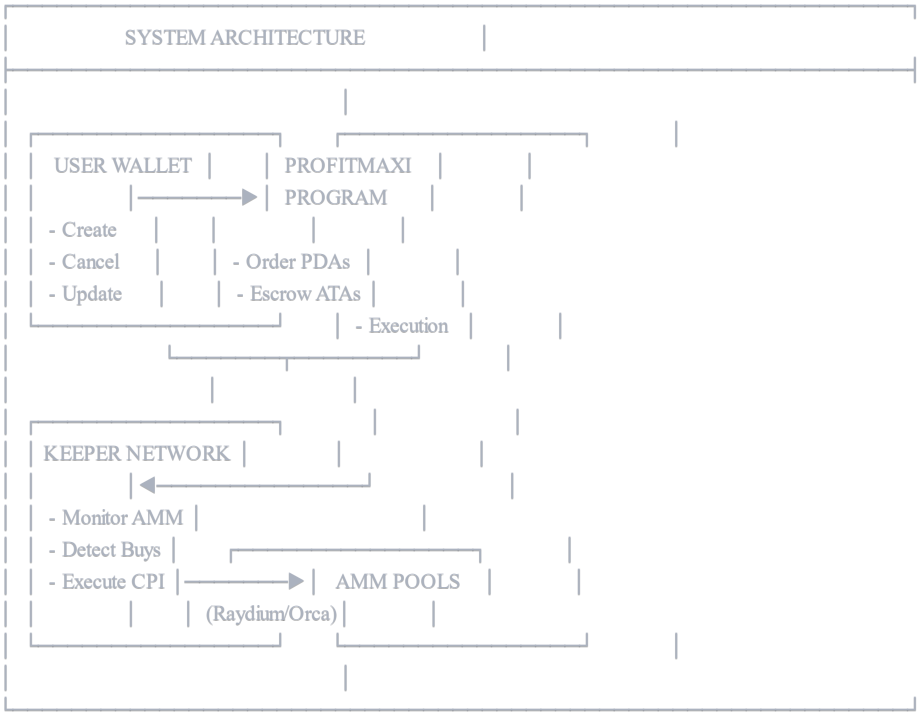
7.3 Execution Mechanics

1. **Trigger Detection:** Keeper monitors AMM for buys ≥ threshold
2. **Shard Calculation:** `sell_amount = min(r × buy_amount, remaining)`
3. **Atomic Execution:** CPI swap from escrow via AMM program
4. **State Update:** Decrement remaining, emit event
5. **Repeat:** Continue until remaining = 0

---

8. Implementation Architecture

8.1 System Components



8.2 On-Chain Program (Anchor)

Account Structure:

rust

```rust
pub struct Order {
    pub owner: Pubkey,          // 32 bytes
    pub token_mint: Pubkey,     // 32 bytes
    pub total_size: u64,        // 8 bytes
    pub remaining: u64,         // 8 bytes
    pub delta_ratio_bps: u16,   // 2 bytes
    pub min_threshold: u64,     // 8 bytes
    pub escrow_ata: Pubkey,     // 32 bytes
    pub status: OrderStatus,    // 1 byte
    pub bump: u8,               // 1 byte
}
```

Instructions:

- `create_order`: Initialize order, escrow tokens
- `execute_shard`: Keeper-triggered partial fill
- `cancel_order`: Return escrowed tokens
- `update_order`: Modify parameters

8.3 Keeper Network

Keepers monitor AMM pools and trigger execution. Multiple strategies:

1. **Websocket Monitoring:** Real-time account subscriptions
2. **Log Parsing:** Subscribe to AMM program logs
3. **Geyser Plugins:** Direct node access (institutional)
4. **Clockwork:** Decentralized on-chain automation

8.4 MEV Protection

Execution submitted via **Jito bundles** to prevent:

- Front-running by other keepers
- Sandwich attacks on ProfitMaxi sells

- Failed transaction spam

---

9. Economic Analysis

9.1 Fee Structure

Fee Type	Amount	Recipient
Order Creation	0.01 SOL	Protocol
Execution (per shard)	0.1% of sell	Keeper
Cancellation	Free	-

9.2 Keeper Economics

Keeper profitability depends on:

- **Volume:** Higher AMM volume = more triggers
- **Competition:** More keepers = faster execution but lower individual rewards
- **Gas Costs:** ~5000 lamports per execution

Expected Keeper Revenue:

$$\text{Revenue} = \sum_i 0.001 \times \text{Sell}_i - \text{Gas}_i$$

Revenue = i∑ 0.001 × Selli − Gasi

9.3 Protocol Revenue

At scale, protocol revenue from creation fees:

$$\text{Annual Revenue} = N_{\text{orders}} \times 0.01 \text{ SOL}$$

Annual Revenue = Norders × 0.01 SOL

9.4 User Value Proposition

Without ProfitMaxi:

- 50 SOL position exit → ~25% price impact → 12.5 SOL lost to slippage

With ProfitMaxi (r=1.0):

- 50 SOL position exit → ~1% price impact → 0.5 SOL lost
- **Savings:** 12 SOL (96% reduction in impact)

---

10. Security Considerations

10.1 Smart Contract Risks

Risk	Mitigation
Escrow theft	PDA-controlled ATAs, owner-only cancel
Reentrancy	Checks-effects-interactions pattern
Integer overflow	Checked math, Rust safety
Oracle manipulation	No external price oracle dependency

10.2 Economic Attacks

Griefing Attack: Malicious actor sends many small buys to trigger execution.

- *Mitigation:* `min_threshold` parameter filters dust

Keeper Collusion: Keepers front-run or delay execution.

- *Mitigation:* Jito bundles, multiple competing keepers

AMM Manipulation: Attacker manipulates pool before execution.

- *Mitigation:* Atomic execution within same block

10.3 Operational Risks

- **Keeper Liveness:** Orders depend on active keepers
- **Network Congestion:** May delay execution during high load
- **AMM Changes:** Protocol upgrades may break CPI

11. Conclusion

ProfitMaxi introduces a novel order type that addresses a critical gap in decentralized finance: the ability to exit large positions without destroying price. By responding to market demand rather than arbitrary schedules, ProfitMaxi enables:

1. **Stealth Exits:** Minimal chart impact for sellers
2. **Price Support:** Configurable positive drift for projects
3. **Trustless Execution:** Fully on-chain, keeper-incentivized
4. **Retail Accessibility:** Institutional-grade execution for all

The mathematical foundations are sound, empirically validated, and the architecture is implementable with current Solana primitives. We anticipate significant adoption among:

- Project teams managing treasury/team allocations
- Early investors seeking low-impact exits
- Market makers managing inventory
- DAOs liquidating holdings

Future Work

- Multi-pool support (cross-DEX execution)
- Dynamic delta ratio adjustment based on conditions
- Integration with token launch platforms
- Governance token for protocol upgrades

---

12. References

1. Angeris, G., Kao, H. T., Chiang, R., Noyes, C., & Chitra, T. (2020). An analysis of Uniswap markets.
2. Adams, H., Zinsmeister, N., Salem, M., Keefer, R., & Robinson, D. (2021). Uniswap v3 Core.
3. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., ... & Juels, A. (2020). Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability.
4. Moallemi, C. C., & Roughgarden, T. (2023). Optimal Execution in Decentralized Finance.
5. Park, A. (2021). The Conceptual Flaws of Constant Product Automated Market Making.

Appendix A: Simulation Code

Full simulation code available at: [GitHub Repository]

Appendix B: Anchor Program IDL

Complete program interface available at: [GitHub Repository]

Appendix C: Keeper Reference Implementation

TypeScript keeper service reference at: [GitHub Repository]

Contact:
d3f4ult
Mezzanine DAO
[Website] | [Twitter] | [Discord]
