Minimalisasi Slippage pada Transaksi Cryptocurrency di Bursa Cryptocurrency Terdesentralisasi dengan Algoritma A-Star

Muhammad Abdul Aziz Ghazali - 13521128
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : alilo.ghazali@gmail.com

***Abstract*—Penelitian ini menyelidiki penggunaan algoritma A-star dalam mengurangi slippage pada perdagangan cryptocurrency di exchange terdesentralisasi. Fokus penelitian ini adalah memecahkan masalah likuiditas rendah yang menyebabkan slippage yang tinggi. Likuiditas rendah dapat mempengaruhi harga aset saat permintaan tinggi. Tujuan penelitian ini adalah mengoptimalkan likuiditas pada exchange terdesentralisasi dengan menggunakan algoritma A-star. Kami merancang algoritma ini untuk memilih liquidity pool dengan likuiditas yang memadai, sehingga mengurangi dampak slippage pada transaksi cryptocurrency. Diharapkan penelitian ini dapat meningkatkan pengalaman perdagangan pengguna dan memberikan kontribusi dalam mengatasi slippage dengan mempertimbangkan faktor likuiditas. Hasil penelitian ini dapat membantu meningkatkan efisiensi dan keandalan perdagangan cryptocurrency di platform terdesentralisasi.**

Keywords—Cryptocurrency, Blockchain, Decentralized Exchange (DEX), A-star algorithm

I. PENDAHULUAN

A. Latar Belakang

Pada era digital saat ini, perdagangan aset digital atau cryptocurrency telah menjadi salah satu bentuk investasi yang populer dan berkembang pesat. Exchange terdesentralisasi, yang beroperasi di atas teknologi blockchain, telah memungkinkan pengguna untuk melakukan transaksi langsung tanpa perantara pusat atau otoritas terpusat. Namun, dalam konteks perdagangan cryptocurrency, ada tantangan khusus yang harus dihadapi, salah satunya adalah slippage.

Slippage adalah fenomena ketika harga aset yang diinginkan oleh pengguna tidak sama dengan harga yang dieksekusi saat melakukan transaksi. Dalam exchange terdesentralisasi, slippage dapat terjadi karena beberapa faktor, seperti likuiditas yang rendah, volatilitas pasar, dan ketidaktepatan pelaksanaan transaksi. Slippage dapat mengakibatkan pengguna mengalami kerugian finansial dan merusak pengalaman perdagangan mereka.

Untuk mengatasi permasalahan slippage, diperlukan pendekatan yang efektif dalam meminimalkan slippage dalam transaksi di exchange terdesentralisasi. Dalam makalah ini, kami akan menyelidiki penggunaan algoritma A-star untuk mencari jalur transaksi antara sepasang *cryptocurrency* dengan tujuan mengurangi slippage khususnya berhubungan erat dengan tingkat likuiditas. Algoritma pathfinding, seperti A* (A-star), telah terbukti efektif dalam menemukan jalur terpendek atau teroptimal dalam berbagai konteks, dan kami akan mengadaptasinya ke dalam lingkungan DEX.

B. Permasalahan

Dalam konteks exchange terdesentralisasi, likuiditas yang rendah menjadi salah satu permasalahan utama yang berkontribusi pada slippage yang tinggi dalam transaksi. Oleh karena itu, dalam penelitian ini, fokus utama kami adalah bagaimana algoritma pathfinding dapat membantu meminimalkan dampaknya pada slippage.

Beberapa permasalahan terkait likuiditas yang rendah dalam exchange terdesentralisasi yang ingin kami atasi meliputi:

a. Keterbatasan Likuiditas:
Exchange terdesentralisasi seringkali menghadapi tantangan dalam mencapai likuiditas yang memadai, terutama pada aset-aset dengan volume perdagangan yang rendah. Keterbatasan likuiditas dapat menyebabkan slippage yang signifikan, karena ketika likuiditas rendah, permintaan pembelian atau penjualan aset tertentu dapat menggerakkan harga secara signifikan. Oleh karena itu, diperlukan pendekatan yang dapat mengoptimalkan likuiditas pada exchange terdesentralisasi untuk mengurangi dampak slippage pada transaksi.

b. Identifikasi Rute dengan Likuiditas Optimal:
Dalam kondisi likuiditas yang rendah, penting untuk mengidentifikasi rute transaksi dengan likuiditas optimal untuk meminimalkan slippage. Ini melibatkan menemukan jalur yang melibatkan aset-aset dengan likuiditas yang lebih tinggi atau melibatkan kebijakan routing yang cerdas untuk memilih liquidity pool yang memiliki likuiditas yang memadai.

Dalam makalah sederhana ini, kami akan merancang algoritma A-star untuk mengoptimalkan rute transaksi dengan mempertimbangkan faktor likuiditas aset yang terlibat.

C. Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk mengembangkan dan menerapkan algoritma A-Star (A*) dalam konteks bursa cryptocurrency terdesentralisasi untuk meminimalkan slippage dengan berfokus pada permasalahan slippage akibat likuiditas yang rendah. Algoritma A-Star akan digunakan untuk mencari jalur transaksi terbaik yang mempertimbangkan faktor likuiditas aset *cryptocurrency* yang terlibat.

D. Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan kontribusi sebagai berikut :

a. Malam mengurangi dampak slippage pada transaksi cryptocurrency di bursa cryptocurrency terdesentralisasi
b. Meningkatkan pengalaman perdagangan pengguna sehingga apa yang pengguna dapat adalah apa yang pengguna beli.
c. Memberikan ruang bagi peneliti yang ingin mengatasi permasalahan *slippage* lebih lanjut dengan memperhatikan faktor-faktor selain likuiditas.

E. Ruang Lingkup Penelitian

Ruang lingkup makalah ini mencakup hal-hal berikut:

a. Bursa Cryptocurrency Terdesentralisasi:
Penelitian ini akan berfokus pada bursa cryptocurrency terdesentralisasi yang beroperasi di atas teknologi blockchain. Kami akan memilih beberapa bursa cryptocurrency terdesentralisasi yang representatif untuk melakukan analisis algoritma A-Star dalam konteks penelitian ini.

b. Slippage Akibat Likuiditas Rendah:
Fokus utama penelitian ini adalah mengatasi slippage yang disebabkan oleh likuiditas rendah dalam transaksi cryptocurrency.

c. Algoritma A-Star:
Penelitian ini akan mengimplementasikan algoritma A-Star untuk mencari jalur transaksi dengan likuiditas optimal. Kami akan mengadaptasi algoritma A-Star agar sesuai dengan konteks bursa cryptocurrency terdesentralisasi dan meminimalkan dampak slippage akibat likuiditas rendah.

II. LANDASAN TEORI

A. Cryptocurrency

Cryptocurrency adalah bentuk aset digital yang menggunakan teknologi kriptografi untuk mengamankan transaksi dan mengendalikan penciptaan unit baru. Cryptocurrency beroperasi di atas teknologi blockchain, yang merupakan ledger terdistribusi yang mencatat semua transaksi yang terjadi. Cryptocurrency memiliki karakteristik seperti desentralisasi, anonimitas, dan keamanan yang tinggi. Beberapa contoh cryptocurrency terkenal adalah Bitcoin, Ethereum, dan Ripple.

B. Decentralized Exchange

Exchange terdesentralisasi, juga dikenal sebagai Decentralized Exchange (DEX), adalah platform perdagangan cryptocurrency yang beroperasi tanpa kehadiran otoritas pusat. Pada exchange terdesentralisasi, transaksi cryptocurrency terjadi secara langsung antara pengguna tanpa melalui perantara. Exchange terdesentralisasi berjalan di atas teknologi blockchain, yang memastikan keamanan dan transparansi dalam proses perdagangan. Kelebihan dari exchange terdesentralisasi adalah kontrol pengguna terhadap aset mereka, kebebasan dari ketergantungan pada entitas pusat, dan keamanan yang tinggi. Namun, exchange terdesentralisasi juga memiliki kelemahan seperti likuiditas yang rendah dan masalah slippage.

C. Slippage

Slippage adalah fenomena ketika harga aset yang diinginkan oleh pengguna tidak sama dengan harga yang dieksekusi saat melakukan transaksi. Dalam konteks exchange terdesentralisasi, slippage dapat terjadi karena beberapa faktor, seperti likuiditas yang rendah, volatilitas pasar, dan ketidaktepatan pelaksanaan transaksi. Slippage dapat menyebabkan pengguna mengalami kerugian finansial dan merusak pengalaman perdagangan mereka.

D. Liquidity Pool

Liquidity pool merupakan konsep yang terkait dengan likuiditas dalam konteks exchange terdesentralisasi. Liquidity pool adalah kumpulan aset yang tersedia untuk diperdagangkan di suatu platform atau protokol terdesentralisasi. Liquidity pool berfungsi untuk menyediakan likuiditas dalam pasar dengan cara mengumpulkan dana dari para penyedia likuiditas (liquidity providers) yang ingin meminjamkan aset mereka untuk keperluan perdagangan. Liquidity pool biasanya terdiri dari pasangan aset yang berbeda, seperti ETH/USDT, BTC/USDC, atau aset-aset lain yang diperdagangkan di bursa tersebut.

Liquidity pool memiliki peran penting dalam mengatasi masalah likuiditas rendah dalam exchange terdesentralisasi. Dengan adanya liquidity pool, para trader dapat melakukan transaksi dengan likuiditas yang lebih baik, bahkan pada aset dengan volume perdagangan yang rendah. Liquidity pool menyediakan pasangan aset dengan likuiditas yang memadai, sehingga trader dapat melakukan pembelian atau penjualan dengan harga yang lebih optimal. Liquidity pool juga memungkinkan trader untuk memperoleh likuiditas secara instan, mengurangi kemungkinan slippage yang tinggi.

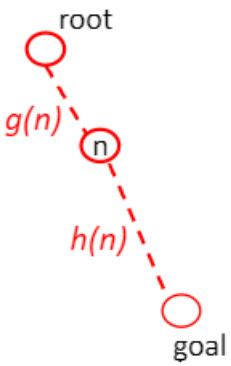
Dalam exchange terdesentralisasi, liquidity pool umumnya didukung oleh protokol terdesentralisasi, seperti Automated Market Maker (AMM) atau Constant Function Market Maker (CFMM). Protokol ini mengatur mekanisme pertukaran aset dalam liquidity pool dengan menggunakan algoritma yang ditentukan. AMM memungkinkan trader untuk memasukkan atau menarik likuiditas ke dalam liquidity pool dengan menyetorkan aset mereka ke dalam smart contract. Dalam AMM, harga aset ditentukan berdasarkan perbandingan keseimbangan aset di dalam liquidity pool.

E. Algoritma A-star

Algoritma A-Star, atau dikenal juga sebagai A*, adalah algoritma pencarian jalur atau pathfinding yang digunakan untuk mencari jalur terpendek atau teroptimal antara dua titik dalam graf atau ruang pencarian. Algoritma ini sering digunakan dalam berbagai konteks, termasuk navigasi robot, game, dan optimisasi rute.

Algoritma A-Star menggabungkan dua fungsi heuristik untuk mencari jalur terbaik dengan n adalah suatu simpul status dalam rute yang bisa dibentuk oleh permasalahan:

a. Fungsi g(n) adalah fungsi yang menghitung biaya sejauh ini untuk mencapai titik saat ini (n) dari titik awal.
b. Fungsi h(n) adalah fungsi heuristik yang memperkirakan biaya yang tersisa untuk mencapai titik akhir dari titik saat ini (n). Fungsi heuristik ini memberikan perkiraan jarak atau biaya yang tersisa berdasarkan estimasi yang diberikan. Fungsi heuristik tidak boleh *overestimate* biaya asli dari n ke *state goal*.



Ilustrasi 2.5.1 : Visualisasi Fungsi g(n) dan h(n)
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf)

Algoritma A-Star menggunakan perkiraan total biaya untuk mengevaluasi jalur dengan persamaan:

$$f(n) = g(n) + h(n)$$

Dengan memanfaatkan fungsi heuristik, algoritma ini dapat secara efisien mencari jalur terpendek dengan meminimalkan jumlah langkah yang diperlukan yaitu tidak melakukan *expand* pada jalur yang sudah mahal biayanya.

Algoritma A-Star menggunakan pendekatan heuristik untuk mencari jalur terpendek. Prinsip kerjanya melibatkan eksplorasi graf secara sistematis dengan mempertimbangkan biaya aktual dan estimasi biaya yang tersisa. Berikut adalah langkah-langkah umum dalam algoritma A-Star:

1. Inisialisasi:
   - Tentukan titik awal dan titik akhir dalam graf.
   - Tetapkan biaya awal untuk titik awal dan biaya estimasi untuk mencapai tujuan.
2. Buat dua himpunan:
   - Open Set: Himpunan node yang sedang dieksplorasi. Awalnya hanya berisi titik awal.
   - Closed Set: Himpunan node yang sudah dieksplorasi.
3. Loop hingga Open Set kosong atau titik akhir ditemukan:
   - Pilih node dengan biaya terendah dari Open Set. Inilah node yang akan dieksplorasi selanjutnya.
   - Pindahkan node yang dipilih dari Open Set ke Closed Set.
   - Periksa tetangga-tetangga yang terhubung dengan node saat ini:
     - Perbarui biaya aktual dari titik awal ke tetangga jika ditemukan biaya yang lebih rendah.
     - Hitung estimasi biaya dari tetangga ke titik akhir menggunakan heuristik yang sesuai.
     - Tambahkan tetangga ke Open Set jika belum ada di dalamnya.
4. Jika Open Set kosong, jalur tidak ditemukan. Jika titik akhir ditemukan dalam Closed Set, jalur terpendek ditemukan.

Berikut adalah *pseudocode* dari algoritma A-star :

```
function A_Star(start, goal):
    openSet := {start}    // himpunan node yang belum dieksplorasi
    closedSet := {}       // himpunan node yang sudah dieksplorasi

    // Inisialisasi g(n), h(n), dan f(n) untuk node awal
    start.g := 0
    start.h := heuristic(start, goal)
    start.f := start.g + start.h

    while openSet is not empty:
        current := node dengan f(n) terendah di dalam openSet

        if current = goal:
            return reconstructPath(current)

        remove current dari openSet
        add current ke closedSet

        for each neighbor in current.neighbors:
            if neighbor in closedSet:
                continue

            tentative_g := current.g +
```

```
cost(current, neighbor)

            if neighbor not in openSet or
tentative_g < neighbor.g:
                neighbor.g := tentative_g
                neighbor.h                :=
heuristic(neighbor, goal)
                neighbor.f  :=  neighbor.g
+ neighbor.h
                neighbor.parent           :=
current

                if    neighbor    not    in
openSet:
                    add     neighbor     ke
openSet

    return failure

function reconstructPath(node):
    path := [node]

    while node.parent is not null:
        node := node.parent
        add node ke path

    return path
```

Pada algoritma di atas, **start** dan **goal** adalah node awal dan tujuan dalam jaringan yang direpresentasikan. Setiap node memiliki properti seperti **g** (biaya sejauh ini dari node awal), **h** (heuristik perkiraan biaya ke tujuan), **f** (fungsi nilai total yang menggabungkan biaya sejauh ini dan perkiraan biaya ke tujuan), dan **parent** (node sebelumnya dalam jalur terpendek).

Algoritma A* mencari jalur terpendek dari **start** ke **goal** dengan mempertimbangkan biaya sejauh ini (**g**), perkiraan biaya ke tujuan (**h**), dan fungsi nilai total (**f**). Algoritma secara berulang memilih node dengan **f(n)** terendah dari **openSet**, mengeksplorasi tetangga-tetangganya, dan memperbarui biaya dan nilai total jika jalur baru lebih baik. Algoritma berhenti saat mencapai tujuan atau ketika tidak ada jalur yang tersedia.

Fungsi **reconstructPath(node)** digunakan untuk mengembalikan jalur yang ditemukan dari **node** tujuan kembali ke **start**.

III. Analisis

Karena ada ratusan cryptocurrency yang ada di suatu DEX, maka untuk kemudahan testing hanya akan diambil beberapa pasang *cryptocurrency*. Berikut adalah data likuiditas yang didapat dari *decentralized exchange* Sushi pada 20 Mei 2023 :

NO	Nama Pasangan Cryptocurrency	Historical Liquidity
1	BTC-ETH	0.8
2	BTC-ADA	0.5
3	ETHADA	0.6
4	ETH-BCH	0.4
5	ADA-BCH	0.7
6	BTC-XRP	0.3
7	ETH-XRP	0.2
8	ADA-XRP	0.6
9	BCH-XRP	0.5
10	BTC-LTC	0.7
11	ETH-LTC	0.3
12	ADA-LTC	0.4
13	BCH-LTC	0.8
14	XRP LTC	0.6

Dengan keterangan nama 'BTC': 'Bitcoin', 'ETH': 'Ethereum', 'ADA': 'Cardano', 'BCH': 'Bitcoin Cash', 'XRP': 'Ripple', 'LTC': 'Litecoin'

Dengan bahasa python, dimodelkan beberapa hal yaitu :

A. Likuiditas

```python
liquidity_data = {
    ('BTC', 'ETH'): 0.8,
    ('BTC', 'ADA'): 0.5,
    ('ETH', 'ADA'): 0.6,
    ('ETH', 'BCH'): 0.4,
    ('ADA', 'BCH'): 0.7,
    ('BTC', 'XRP'): 0.3,
    ('ETH', 'XRP'): 0.2,
    ('ADA', 'XRP'): 0.6,
    ('BCH', 'XRP'): 0.5,
    ('BTC', 'LTC'): 0.7,
    ('ETH', 'LTC'): 0.3,
    ('ADA', 'LTC'): 0.4,
    ('BCH', 'LTC'): 0.8,
    ('XRP', 'LTC'): 0.6,
}
```

Ilustrasi 3.1.1 : Likuiditas Beberapa Cryptocurrency

Likuiditas dimodelkan dengan pari key dan value untuk diakses di main program nantinya.

B. Calculae Cost

```python
def calculate_cost(node, neighbor):
    # Menghitung biaya berdasarkan
likuiditas antar cryptocurrency
```

```
    liquidity = liquidity_data.get((node,
neighbor), liquidity_data.get((neighbor,
node), None))
    if liquidity:
        return 1 / liquidity
    else:
        return float('inf')
```

Ilustrasi 3.2.1 : Perhitungan Cost

Cost dihitung dengan cara sepertidi atas karena cost akan semakin baik jika likuiditas besar. Sehingga likuiditas dan cost berbanding terbalik.

C. Neighbor

```
def get_neighbors(node):
    # Mendapatkan tetangga (neighbor) dari
node
    neighbors = []
    for pair in liquidity_data.keys():
        if node in pair:
            neighbor = pair[0] if pair[1]
== node else pair[1]
            neighbors.append(neighbor)
    return neighbors
```

Ilustrasi 3.3.1 :Fungsi untuk mendapatkan tentangga

Mendapatkan tetangga dari suatu token dilakukan dengan mengiterasi liquidity_data dan mencari dalam key atau valuenya.

D. A-star

```
def a_star(start, end):
    # Menggunakan algoritma A* untuk
mencari jalur terpendek

    # Inisialisasi nilai awal
    open_set = []
    closed_set = set()

    g_score = {start: 0}
    f_score = {start: calculate_cost(start,
end)}

    heapq.heappush(open_set,
(f_score[start], start))

    while open_set:
        current =
heapq.heappop(open_set)[1]

        if current == end:
            return
reconstruct_path(came_from, end)

        closed_set.add(current)

        for neighbor in
get_neighbors(current):
            if neighbor in closed_set:
                continue

            tentative_g_score =
g_score[current] + calculate_cost(current,
neighbor)

            if neighbor not in g_score or
tentative_g_score < g_score[neighbor]:
                came_from[neighbor] =
current
                g_score[neighbor] =
tentative_g_score
                f_score[neighbor] =
g_score[neighbor] +
calculate_cost(neighbor, end)
                heapq.heappush(open_set,
(f_score[neighbor], neighbor))

    return None
```

Ilustrasi 3.4.1 :Fungsi A-star dengan dasar pseudocode yang ada di bab sebelumnya

Algoritma A-star untuk mencari rute terbaik antara pasangan token ditentukan dengan nilai g(n) dan h(n) yang sesuai heuristik. Disini heuritstik selalu admissible karena nilainya tidak akan pernah overestimate jarak asli ke goal.

E. Path

```
def reconstruct_path(came_from, current):
    # Membangun jalur dari start ke current
menggunakan came_from

    path = [current]
    while current in came_from:
        current = came_from[current]
        path.insert(0, current)
    return path
```

Ilustrasi 3.5.1 :Fungsi untuk mendapatkan path lengkap dengan merekonstrtuksinya

Fungsi ini diperlukan sebagai fungsi bantuan dalam membuat rute dalam fungsi A-star

F. Main Testing

```
# Testing
start_node = 'BTC'
end_node = 'XRP'
came_from = {}
path = a_star(start_node, end_node)

if path:
    print(f"Jalur dari {start_node} ke
{end_node}: {path}")
else:
    print(f"Tidak ada jalur yang tersedia
dari {start_node} ke {end_node}")
```

Ilustrasi 3.6.1 : Testing untuk pasangan BTC dan XRP

Akan digunakan main program untuk mencari rute dari kedua pasang token ini. Jika ditemukan akan ditampilkan hasilnya dan jika tidak ada rute yang bisa didapat maka akan menampilkan pesan tidak ada jalur yang tersedia.

G. Hasil

Dengan menjalankan program tersebut pada lingkungan laptop kami di Visual Studio Code, didapat hasil berikut

```
[Running] python -u "c:\Users\alilo\OneDrive\Documents\KULIAH\Tingkat
2\Stima\Makalah\pathliq.py"
Jalur dari BTC ke XRP: ['BTC', 'LTC', 'XRP']
```

Sehingga, apabila ingin mendapatkan slippage yang paling minimum, dalam melakukan pembelian token di Sushi pada keadaan pasar tersebut adalah dengan membeli LTC dengan BTC. Kemudian membeli XRP dengan LTC.

IV. KESIMPULAN

Didapatkan bahwa untuk membeli suatu token dengan token lain tidak selalu harus langsung melakukan pembelian. Terkadang terjadi slippage yang signifikann sehingga membuat apa yang dibeli tidak sesuai dengan keinginan *trader*. Ini disebabkan karena faktor algoritma AMM pada suatu *liquidity poolI,* kedalaman book order, maupun volitalitas pasar. Permsalahan ini dapat diatasi dengan menghitung rute yang diperlukan untuk meminimalkan slippage namun harus didapatkan dahulu data likuiditas yang menggambarkan pasar beberapa pasangan token yang bisa dibeli.

UCAPAN TERIMA KASIH

Penulis sangat berterima kasih kepada seluruh pihak yang secara langsung maupun tidak langsung dalam membantu penulis menyelesaikan makalah ini. Terimakasih sebesar-besarnya kepada Ibu Dosen Dr.Nur Ulfa Maulidevi, ST,M.Sc. selaku pengampu kelas dalam mata kuliah Strategi Algoritma IF2211. Penulis tidak akan bisa menyelesaikan makalah ini tanpa dukungan dan doa dari teman dan keluarga penulis.

REFERENSI

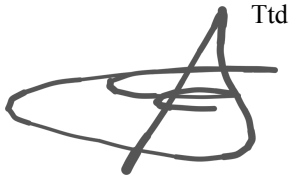
[1] Munir, Rinaldi. (2021). *Route Planning (Bagian 2).*https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf . Diakses pada 21 Mei 2023)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

Ttd



Muhammad Abdul Aziz Ghazali

13521128