EVE Astar: A system-pathfinder solver

Javier Martín Pizarro

April 10, 2025

Abstract

This paper aims to analyze and replicate the pathfinding algorithm used in EVE Online for navigating between star systems. It explores the underlying algorithm, the heuristics applied, and their impact on route optimization. Additionally, the paper discusses the advantages and limitations of this approach in the context of in-game travel.

Keywords: A* algorithm, pathfinding, heuristics

1 Overview

Context EVE Online is considered one of the biggest spatial MMORPG, with more than 7500 systems to travel to. All those systems are connected between each other, conforming a graph $G(V, E)$. Players can travel through different systems following a determined path that can be calculated using an internal tool - a pathfinding solver.

This solver has several use cases, where we can highlight the following:

- Travel from A to B using the **shortest and safest path**.
- Travel from A to B using the **insecurest path**.
- Travel from A to B using the **shortest path**.

Each system has a CONCORD security status, in the interval $[-1.0, 1.0]$. Depending on the system's security, if you are attacked by other players, CONCORD Security will defend you sooner or later - or not. It is possible to divide the space in three categorical variables depending on the security:

- High security space (HS): $0.5 \leq sec \leq 1.0$. CONCORD security will spawn ships for defending your ship. Response time varies depending on the security, the higher the sooner they will appear. Capsuleers cannot freely engage other capsuleers without consequences.
- Low secutity space (LS): $0.1 \leq sec \leq 0.4$. CONCORD security won't spawn ships, but CONCORD turrets will defend your ship. Capsuleers may freely engage other capsuleers.
- Null security space (NS): $-1.0 \leq sec \leq 0.0$. Wormholes are included here. CONCORD does not exist in these systems. Capsuleers may freely engage other capsuleers.

Notice that wormholes connection varies dinamically, depending on *rolling* and external factors. For this reason, they are not considered in the study.

Definition Jump Occurs when the player travels between two systems. In a Dijkstra algorithm, it can be considered as a cost, where $c = 1$.

When hauling items between two points, most players decide to use the shortest and safest path possible, in order to avoid being ganked and reducing the number of jumps.

2 The Problem

It is possible to categorize the problem as a graph problem $G(V, E)$, where each system V is connected through several gates E to another system V'. This route is bidirectional, where travelling $A \leftrightarrow B$ is possible.

Assume that the capsuleer will always want to travel using the **shortest and safest** route possible.

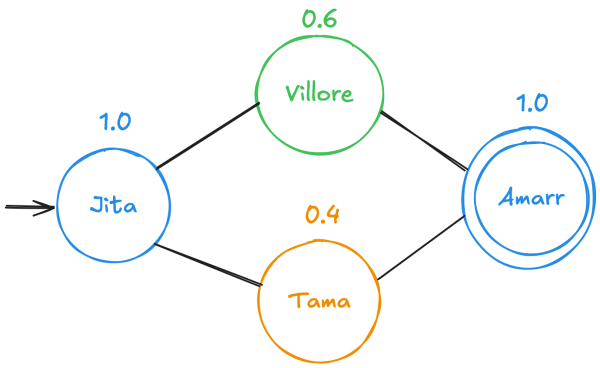

Figure 1: An example of the problem

The previous image shows the simplest case possible, where the user wants to travel from Jita to Amarr. It has two possible paths: either choosing Tama (with a lower security status) or choosing Villore (with a higher security status). The obvious solution is $Jita \rightarrow Villore \rightarrow Amarr$.

It is clearly seen that there is no possible way of calculating and optimal solution using greedy algorithms. For this case, the best algorithm to solve the problem is A^*.

A^* - and its variations - is one of the standards in computer science due to its optimality, completeness and efficiency.

The major drawback of it is considered to be its own spatial complexity $O(b^d)$, where b is the branching factor and d is the depthning factor (more information is given in the subsection 2.2).

2.1 Problem Input

The input of this problem is:

1. A graph $G(V, E)$. The vertices of the graph are possible locations for the capsuleer, and the edges are the possible transitions between locations.
2. Origin system. The system where the user starts its route.
3. Destination system. The system where the user ends its route.

2.2 Actions

Between successive time points, the user can perform a *move_to(A, B)* action iff exists an edge that connects both nodes. The current location of the capsuleer goes from $A \rightarrow B$.

Because of the complexity of the problem, and definition of the possible actions, the branching factor $b = n$, where n are the adjacent vertices of V.

The depthning factor $d = V$, where V is the total number of vertices that conform the graph $G(V, E)$.

2.3 Constraints

Knowing this, it is possible to define a the main points of the problem:

Lemma The capsuleer should avoid, if possible, any LS or NS systems, even if that means that the trip must be extended n jumps.

Lemma While travelling through HS, the capsuleer should not take into consideration the security of the system, as they all are considered equally safe.

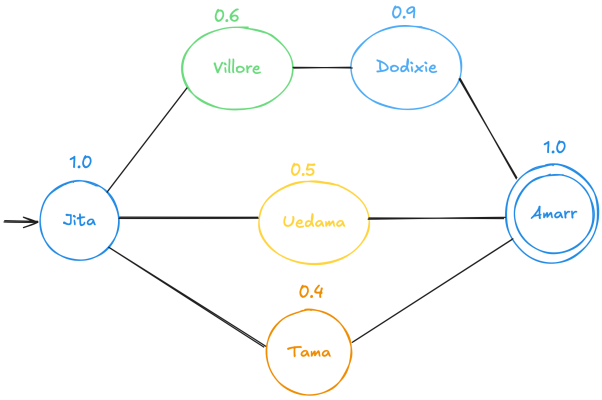

Figure 2: In this image, three possible paths are considered.

In the previous figure, altough the safest route is $Villore \rightarrow Dodixie$, it is not the shortest. And assuming that every HS system is equally important (from a security status point of view), it is necessary to consider another function (heuristics are discussed in subsection 2.4).

2.4 Heuristics

A^* is a best-first algorithm. This means that after expanding a node, it will take the most promising next node. Then, it is possible to define a function $f(n)$ such as:

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

- $g(n)$ is the **cost function** or accumulated cost function from origin to the current node. In this case, it is possible to calculate it using the Dijkstra algorithm.
- $h(n)$ is the heuristic function calculated for each node.

2.4.1 The $h(n)$ function

This fuction is calculated taking into account two possible parameters:

- The distance from the current node N to the destination D.
- The security status of the current system.

The system status plays an important role: depending on the value, a penalty factor may be applied in order to increase the cost C_N. For arbitrary reasons, the penalty in considered as $P = 10000$.

The shortest path r from current node N to the destination D is also considered - and calculated - using Dijkstra algorithm.

Therefore:

$$h(n) = \begin{cases} (1 - \text{sec\_status}) + \text{r}, & \text{if sec\_status} \geq 0.5 \\ (1 - \text{sec\_status}) + (\text{r} \cdot \text{P}), & \text{otherwise} \end{cases}$$

2.5 A*

The following pseudocode(2.5) adapts the A^* algorithm into the pathfinder solver.

Algorithm 1 EVE Astar

Input: Graph G, origin node *origin*, objective node *goal*
Salida: Optimal path from *origin* to *goal*

Open $\leftarrow \{origin\}$
Closed $\leftarrow \emptyset$
$g(origin) \leftarrow 0$
$f(origin) \leftarrow g(origin) + h(origin)$
Predecesor $\leftarrow \emptyset$
while Open $\neq \emptyset$ **do**
    $n \leftarrow$ node in Open with the lowest $f(n)$
    if $n = goal$ **then**
        return n
    end if
    Move n from Open to Close
    for each neighbor m of n **do**
        if $m \in$ Closed **then**
            continue
        end if
        $tentative\_g \leftarrow g(n) + d(n, m)$
        if $m \notin$ Open **or** $tentative\_g < g(m)$ **then**
            Predecesor$[m] \leftarrow n$
            $g(m) \leftarrow tentative\_g$
            $f(m) \leftarrow g(m) + h(m)$
            if $m \notin$ Open **then**
                Add m to Open
            end if
        end if
    end for
end while
return "No possible path"

2.6 Problems encountered

This A^* is not deterministic when executing $Origin - Destination$ and $Destination - Origin$. This is caused due to the expansion order of the nodes. In order to solve this, the A^* is run twice (for both possible cases). If the second possible path has a shorter length and less systems with a lower security, that path will be considered as the best one.

However, this is known to be algorithmically non-optimal. Hence, a new solution is proposed:

Bidirecctional search Starting from the origin and from the destination at the same time, it should be possible to not only find the shortest and safest path in one iteration (instead of two), but also it would be computationally quicker. The algorithm would be complete and admissible, as the heuristic is symetric for both sides.