Finding paths problem and solution

Introduction

People like travels. Sometimes they need to reach some point in the world because they are going to vacations, sometimes this is a business case. People go by train, use ships, use flying machines to reach their destination. Not alway you can take single transport vehicle and go to the destination point from your current location. Often you need to use multiple Transfers between intermediate Points to reach your destination Point.

The problem

The well known problem: finding path between two points, where you have a set of transfers between some of the points in the set. But the problem is even harder when you must consider timings: when each transfer starts and when it ends, when you have to reach the destination point.

If you have lots for Points and Transfers, it is a large amount of data. The large amount of data needs databases to store them into. But if you use simple sql's to find a Path using multiple Transfers using intermediate Points to your destination Point, you are going to **fail**, because there is **no good default solution** to create such a SQL index to work your sql fast enough. The problem is that when you join Transfers using time intervals like

(sourcePoint.arrivalTime < intermediatePoint.departureTime and sourcePoint.arrivalTime >

DATE_ADD(destinationPoint.deparatureTime, INTERVAL 8 HOUR)), there is **no way to crate a good table index** to handle a such condition.

The solution

The solution comes to use discrete time intervals, that I call c**lustering strategy** in my example application.

I will not go to details here, but I provide test results using **clustering strategy** and a **simple sql join strategy** to find travel Pahts.

There are two databases with a generated data to test strategies, mentioned above: **test_normal** and **test_complete**.

Each database contains about 1000 generated Points, and about 1000000 generated transfers for a time period from 2027-01-01 to 2027-12-01.

Points are alligned to a 30x30 grid square, where distance between two neigbour Points is 6000 units. There are periodic transfers between each two neigbour points at 8 directions (up, down, left, right and 4 diagonal directions). This is how **test_normal** database data looks like.

In the **test_complete** database points of coordinate X=0, are treated as HUB points, and have additional travels with **each** other HUB point. Also there are additional travels with each HUB point and remaining points, whose Y coordinate is the same as a HUB point's Y coordinate.

This way you should always find a Path if you use 4 transfers between any two Points in the **test_complete** database.

In the **test_normal** database you will find a Path between two pointas A and B if the distance between them is equal or less Transfer amount multiplied by 6000.

Maximal connection hours should be considered between intermediate points, when you expect to find a Path. The longer connection period between transfers, the higher possibility to find a Path, but for the traveler it is less comfortable to wait longer in the intermediate Point to continue his travel.

Also bigger connection periods makes bigger load on the database when the Path finding algorithm is running.

Summary experiment results

When you use 2 Transfers (one time of transport vehicle cahnge) for finding Path, simple algorithm does well running max 0.17 s ; but if you will try to find a Paht with 3 Transfers (two times of transport vehicle change), running sql always timeouts (longer than 15s).

Using clustering strategy Paths with transfers count 4, are found efficiently **faster than by 1 second**.

Paths with transfers count 5 and connection time 32 hours are found in **4 to 5 s** in database **test_normal**. And in **3 to 7s** in database test_complete.

The experimental code is deployed at http://darbelis.eu:7080, you may try to search Paths by yourself. The long running queries are interrupted by the DB statement config if they reach running time 30s (15s if the Transfers count <= 3).

The detailed experiments data are collected in the separate files.

2025-12-22