Распределенный алгоритм поиска траекторий-компаньонов

Соколов А.П.[1], Алисейчик П.А.[2], Моисеев С.В.[3]

В работе рассматривается задача построения распределенного алгоритма поиска траекторий-компаньонов. Под траекториями-компаньонами понимается пара траекторий такая, что обе траектории обладают участком достаточной длины, на котором оба объекта в каждый момент времени находятся на достаточно близком расстоянии друг от друга.

Рассматривается задача поиска в базе данных траекторий, которая из-за своего размера не может быть размещена и обработана на одной рабочей станции. Подобные задачи принято называть задачами на больших данных или просто big-data-задачами. Определена процедура квантизации траекторий, вводится понятие клеточного индекса, введено определение похожих траекторий, построен эффективный алгоритм поиска похожих траекторий. На базе данного алгоритма построены локальный и распределенный алгоритмы поиска траекторий-компаньонов, получены оценки их сложности.

Ключевые слова: большие данные, анализ пространственно-временных данных, анализ траекторий, поиск траекторий-компаньонов, поиск близких траекторий, анализ трафика.

[1] *Соколов Андрей Павлович* — к.ф.-м.н., м.н.с. каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: sokolov@intsys.msu.ru

Sokolov Andrey Pavlovich — junior scientific researcher, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

[2] *Алисейчик Павел Александрович* — к.ф.-м.н., в.н.с. каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: paa.idea@mail.ru

Aliseychik Pavel Alexandrovich — leading scientific researcher, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

[3] *Моисееев Станислав Владимирович* — выпускник каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: stanislav.moiseev@gmail.com

Moiseev Stanislav Vladimirovich — graduate student, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

1. Введение

В настоящее время сотовыми телефонами пользуются практически все. Более того, на многих автомобилях установлены терминалы, подключенные к сотовым сетям связи. В определенных условиях сотовые телефоны и смартфоны могут формировать ценную информацию о своем местоположении. Эта информация возникает на оборудовании оператора сотовой сети в моменты переключения оборудования пользователя от одной базовой станции к другой. Информация о местоположении пользователя определяется местоположением базовой станции, к которой было осуществлено подключение. Заметим, что данная информация возникает даже при отсутствии в оборудовании пользователя высокоточного навигационного оборудования.

В результате у оператора сотовой сети может быть получена и накоплена информация о местоположении и передвижении всех пользователей. Эта информация может быть использована для решения множества полезных практических задач, таких как: анализ транспортных потоков, определение так называемых транзитных точек, через которые проходит большинство маршрутов, поиск траекторий с заданными свойствами, поиск пар траекторий с взаимными свойствами, моделирование транспортной сети и многие другие.

Одной из интересных задач является задача поиска множества траекторий-компаньонов.

Под траекториями-компаньонами понимается пара траекторий такая, что каждая из траекторий содержит в себе участок достаточной длины, на протяжении которого в каждый момент времени оба объекта находятся на достаточно близком друг от друга расстоянии.

Заметим, что задача поиска всех пар траекторий-компаньонов может быть обобщена на случай поиска множеств траекторий-компаньонов заданной мощности. Алгоритмы, описанные в данной работе, могут быть легко обобщены на этот случай.

Задачи поиска множеств траекторий-компаньонов, осуществляющих совместное перемещение в пространстве, рассматриваются, в частности, в публикациях [1], [2], [3].

Работа [1] посвящена поиску множеств объектов, осуществляющих совместное передвижение. Рассматриваются различные типы поведения таких групп: стадо, конвой и рой. Вводятся понятия «толпы» и «скопления». Далее для данных понятий строятся алгоритмы поиска как для статических баз данных, так и для динамически пополняемых.

В работе [2] рассматривается задача поиска и поддержания в актуальном состоянии множеств траекторий-компаньонов на основе непрерывного потока траекторных данных.

В работе [3] вводятся понятия «рой» и «замкнутый рой». Построен алгоритм для эффективного поиска «замкнутых роев». Получены оценки сложности алгоритма.

Настоящая работа посвящена разработке распределенного алгоритма поиска всех пар траекторий-компаньонов. Распределенный алгоритм описан с использованием операций map/reduce [4], которые широко применяется в современных распределенных вычислительных платформах. Таких как, например, Apache Spark.

2. Определения и постановка задачи

Обозначим $O = \{o^1, o^2, ..., o^n\}$ — множество движущихся объектов, которые присутствуют в базе данных. Далее, $T = \{t_1, t_2, ..., t_m\}$ — множество моментов времени, для которых заданы точки траекторий. Множество T задает единую «временную сетку», в которой задаются положения объектов. Будем полагать, что каждая траектория содержит ровно m точек.

Множество моментов времени из T, начинающееся с t_a и заканчивающееся t_b, будем называть отрезком и обозначать $[t_a, t_b]$.

Траекторией объекта o_i будем называть ломаную, заданную конечной последовательностью точек на плоскости для отрезка времени $[t_1, ..., t_m]$

$$o^i = \left\langle p_1^i, ..., p_m^i \right\rangle .$$

Здесь $p_j^i = \left(x_j^i, y_j^i, t_j^i\right)$, где $x_j^i, y_j^i \in \mathbb{R}$ — координаты объекта o_i в пространстве, t_j^i — момент времени.

Заметим, что в реальных базах данных траектории могут иметь различные моменты времени начала и моменты окончания. Также отметим, что возможны ситуации, когда для какого-то момента времени t_j в траектории для объекта o^i может не быть информации. То есть точка p_j^i может отсутствовать в базе данных. В рамках данной работы мы полагаем, что такие ситуации исключены, то есть данные о положении объектов заданы для всех моментов времени из T.

Для простоты будем отождествлять объекты со своими траекториями. Поэтому под o^i будем понимать как объект, так и его траекторию. Множество всех траекторий будем обозначать O.

Заметим, что на практике положения объектов обычно задаются в географических координатах: широта и долгота. Однако в случае, если все рассматриваемые траектории сосредоточены в относительно небольшой области на поверхности Земли, то можно перейти к рассмотрению некоторого отображения всех точек поверхности Земли на плоскость. Такие отображения обычно называют картографическими проекциями.

Одним из примеров такого отображения является равнопромежуточная проекция [7]. В этом случае переход от географических координат к координатам на плоскости выполняется так. Пусть $p' = (\psi, \phi)$ — точка на поверхности Земли с долготой ψ и широтой ϕ, пусть $p'_r = (\psi_r, \phi_r)$ — некоторая базовая точка, расположенная в интересующей нас области. Обозначим $p = (x, y)$ — образ точки p' на плоскости. Тогда:

$$x = R \cdot \cos(\psi) \cdot (\psi - \psi_r),$$

$$y = R \cdot (\varphi - \varphi_r),$$

где R — средний радиус Земли.

Данное отображение приведено в качестве примера. Дальнейшее изложение принципиально не зависит от выбора той или иной картографической проекции.

Дадим определение траекторий-компаньонов. Рассмотрим $o^i, o^j \in O$ — две траектории. Введем параметры $t_{min}, d_{min} \in \mathbb{R}$. Пару траекторий $\left(o^i, o^j\right)$ будем называть (t_{min}, d_{min})-*компаньонами*, если существуют моменты времени $t_a < t_b$, $t_b - t_a \geq t_{\min}$, $t_a, t_b \in T$, такие что для любого момента времени $t_k \in [t_a, t_b]$ имеет место $d\left(p_k^i, p_k^j\right) \leq d_{\min}$, где d — обозначает евклидово расстояние. Отрезок $[t_a, t_b]$ будем называть совместным участком траекторий-компаньонов.

Параметры t_{min}, d_{min} характеризуют минимальную продолжительность *совместного участка* двух траекторий и максимально допустимое расстояние между объектами на протяжении совместного участка. Данные параметры являются фиксированными и не зависят от выбора объектов.

Далее, для краткости, траектории, являющиеся (t_{min}, d_{min})- *компаньонами*, будем называть просто *траекториями-компаньонами*.

Задача поиска пар траекторий-компаньонов ставится следующим образом: дано множество траекторий $O = \{o^1, o^2, ..., o^n\}$ и значения параметров $t_{\min}, d_{\min} \in \mathbb{R}$, необходимо найти все пары $\left(o^i, o^j\right)$, такие что траектории o^i и o^j являются траекториями-компаньонами.

Дальнейшее изложение построено следующим образом. Сначала мы опишем алгоритм вычисления предиката компаньонства и дадим оценку сложности тривиального алгоритма поиска всех пар компаньонов. Затем мы опишем способ квантизации траекторий, позволяющий перейти от рассмотрения вещественных координат, к целочисленным. Далее будет описан алгоритм построения клеточного индекса — специальной структуры данных для эффективного поиска на множестве траекторий. Далее будет введено понятие похожих траекторий и описан алгоритм

для их быстрого поиска. Затем мы построим локальный алгоритм поиска траекторий-компаньонов и дадим оценку его временной сложности. Затем мы опишем распределенный алгоритм поиска траекторий-компаньонов в терминах операций map/reduce.

3. Алгоритм вычисления предиката компаньонства

Введем предикат компаньонства $r\left(o^u, o^v\right)$, который равен 1, если траектории o^u, o^v являются компаньонами, и 0 - в противном случае.

Алгоритм для вычисления предиката $r\left(o^u, o^v\right)$ описан далее (3.1).

Algorithm 3.1 Вычисление предиката компаньонства — TestPair

function TestPair(o^u,o^v)
  Дано: пара траекторий (o^u, o^v)
  Найти: значение предиката $r\left(o^u, o^v\right)$
  флаг нахождения внутри совместного участка
  $flag \leftarrow 0$
  for j=1,. . . ,m **do**
    if $(flag = 0)\,\&\left(d\left(p_j^u, p_j^v\right) < d_{min}\right)$ **then**
      совместный участок начался
      $flag \leftarrow 1$
    end if
    if flag = 1 **then**
      if $\left(d\left(p_j^u, p_j^v\right) < d_{min}\right)\,\&\left(d\left(p_{j+1}^u, p_{j+1}^v\right) < d_{min}\right)$ **then**
        совместный участок продолжается
        $t_{cmn} \leftarrow t_{cmn} + (t_{j+1} - t_j)$
      end if
      if $(d\left(p_j^u, p_j^v\right) < d_{min})\&(d\left(p_{j+1}^u, p_{j+1}^v\right) \geq d_{min})$ **then**
        совместный участок завершен
        $flag \leftarrow 0$
        проверяем, превышает ли длительность заданный порог
        if $t_{cmn} > t_{min}$ **then**
          return 1
        end if
      end if
    end if
  end for
  return 0
end function

Обозначим за F — временную сложность вычисления предиката $r\left(o^i, o^j\right)$. Из описания алгоритма 3.1 очевидно следует, что $F = O(m)$, где m — число моментов времени, на которых заданы положения объектов.

Тривиальный алгоритм поиска всех пар траекторий-компаньонов на множестве O предполагает перебор всех пар $\left(o^i, o^j\right) \in O^2$ и вычисление для каждой пары
предиката r. Очевидно, что временная сложность такого алгоритма равна $C_n^2 \cdot F = O(m \cdot n^2)$. Заметим, что данная оценка справедлива при одновременном стремлении к бесконечности как числа объектов — n, так и числа точек в траекториях — m.

С практической точки зрения интересна задача построения алгоритма, сложность которого относительно n была бы ниже.

В следующих разделах мы построим алгоритм поиска всех пар компаньонов, который при определенных ограничениях на множество траекторий имеет линейную сложность как по m, так и по n.

4. Квантизация траекторий

В данном разделе описан метод квантизации траекторий. Этот метод предназначен для снижения объема траекторной информации и упрощения ее обработки.

Квантизация траекторий заключается в переходе от рассмотрения точек с вещественными координатами к точкам с целочисленным координатам.

Процедура квантизации основана на разбиении пространства и времени на ячейки фиксированного размера. Обозначим $\Delta_s \in \mathbb{R}$ — размер ячейки в пространстве,
$\Delta_t \in \mathbb{R}$ — размер ячейки во времени.

Пусть $p = (x, y, t)$ — некоторая точка в пространстве-времени. Обозначим
$q\left(p\right) = \left(\hat{x}, \hat{y}, \hat{t}\right)$, где

$$\hat{x} = \left[\frac{x}{\Delta_\mathrm{s}}\right], \hat{y} = \left[\frac{y}{\Delta_\mathrm{s}}\right], \hat{t} = \left[\frac{t}{\Delta_\mathrm{t}}\right], \hat{x}, \hat{y}, \hat{t} \in N.$$

Преобразование $q\left(p\right) = \left(\hat{x}, \hat{y}, \hat{t}\right)$ будем называть (Δ_s, Δ_t)-*квантизацией*. Значения $\left(\hat{x}, \hat{y}, \hat{t}\right)$ представляют собой индексы ячеек пространства-времени по соответствующим координатам.

Переход от вещественных координат к целочисленным индексам ячеек позволяет значительно упростить процедуру сравнения траекторий. Однако, такая процедура приводит к снижению точности описания траектории.

Также при квантизации может существенно сократиться объем памяти, необходимый для хранения траекторных данных.

Если мы обозначим максимальные по модулю значения индексов ячеек по пространству и времени как N_s и N_t, то мы можем совместить все три индекса (два индекса пространства и один индекс времени) в один обобщенный индекс ячейки следующим образом

$$\hat{c} = w\left(\hat{x}, \hat{y}, \hat{t}\right) = \hat{t} \cdot N_s^2 + \hat{y} \cdot N_s + \hat{x}.$$

Таким образом, последовательное применение функций q и w ставит в соответствие каждой точке p ячейку пространства-времени с индексом $\hat{c} = w\left(q\left(p\right)\right)$. В дальнейшем, для краткости будем использовать следующие обозначения:

$$\hat{c}\left(p\right) = w\left(q\left(p\right)\right),$$
$$p' = \left(x', y', t'\right) = z\left(\hat{c}\right),$$
$$x' = x\left(\hat{c}\right), y' = y\left(\hat{c}\right), t' = t\left(\hat{c}\right).$$

Здесь функция $\hat{c}$ ставит в соответствие каждой точке пространства-времени
$p = (x, y, t)$ целочисленный обобщенный индекс ячейки, а функция z каждой ячейке с индексом $\hat{c}$ ставит в соответствие точку $p' = (x', y', t')$, являющуюся геометрическим центром ячейки $\hat{c}$. Функции $x\left(\hat{c}\right), y\left(\hat{c}\right), t\left(\hat{c}\right)$
по индексу ячейки возвращают соответствующие координаты ее геометрического центра.

Отметим также, что переход от индексов по пространству и времени $\hat{x}, \hat{y}, \hat{t}$ к обобщенному индексу $\hat{c}$ полностью обратим:

$$\hat{t}\left(\hat{c}\right) = \hat{c}\, div\, N_s^2;$$
$$\hat{y}\left(\hat{c}\right) = \left(\hat{c}\, mod\, N_s^2\right)\, div\, N_s;$$
$$\hat{x}\left(\hat{c}\right) = \hat{c}\, mod\, N_s.$$

В результате процедуры квантизации, каждой траектории o^i может быть поставлена в соответствие последовательность обобщенных индексов ячеек — $o^i \sim \left(\hat{c}_1^i, \ldots, \hat{c}_m^i\right)$.

Квантизацию траектории o^i будем обозначать так: $\hat{o}^i = Q\left(o^i\right)$.

Отметим, что чем меньше значения параметров $\Delta_s,\ \Delta_t$, тем ближе будет квантизованная траектория к исходной.

Пусть $\hat{c}$ — обобщенный индекс некоторой ячейки. Обозначим

$$D\left(\hat{c}\right) = \left\{w\left(\hat{x}\left(\hat{c}\right) + dx, \hat{y}\left(\hat{c}\right) + dy, \hat{t}\left(\hat{c}\right) + dt\right) | dx, dy, dt \in \{-2, 0, 2\}\right\}$$

множество ячеек, соседних с ячейкой $\hat{c}$.

Заметим, что если

$$\Delta_s = \frac{d_{\min}}{\sqrt{2}},$$

где $d_{\min}$ — максимально допустимое расстояние между компаньонами, то всякие две точки, которые находятся на расстоянии, не превышающем $d_{\min}$, в результате процедуры квантизации гарантированно попадут в соседние ячейки. При этом под соседними ячейками понимаются такие ячейки, у которых индексы по каждой из координат отличаются не более, чем на 2. Далее, будем полагать, что данное ограничение на Δ_s выполнено.

Также будем полагать, что $\Delta_t \ll t_{min}$.

5. Алгоритм построения клеточного индекса

В данном разделе вводится специальная структура данных — *клеточный индекс*, и описывается алгоритм для его построения.

Клеточный индекс для каждой ячейки пространства-времени содержит список объектов, которые находятся в данной ячейке или соседней с ней.

Иными словами, клеточный индекс реализует функцию

$$K\left(\hat{c}\right) = \left\{ i | \exists j \in \{1, ..., m\}, \hat{c} \in D\left(\hat{c}_j^i\right) \right\}$$

Отметим, что на практике данные обычно являются разреженными, т.е. существует значительное количество пространственно-временных ячеек, в которых отсутствуют точки траекторий. В связи с этим, для реализации клеточного индекса целесообразно использовать контейнер типа Ключ-Значение. Ключом поиска в этом массиве будет обобщенный индекс ячейки, а значением — множество идентификаторов объектов, которые посещали данную ячейку или соседние с ней.

Ниже приведено описание алгоритма построения клеточного индекса (5.1).

6. Похожие траектории и алгоритм их поиска

Положим $o^i, o^j \in O$ — две траектории. Далее, рассмотрим последовательности ячеек, полученные в результате квантизации:

$$\hat{o}^i = \left(\hat{c}_1^i, \ldots, \hat{c}_m^i\right),$$

Algorithm 5.1 Построение клеточного индекса — BuildIndex

function BUILDINDEX($\hat{O} = \{\hat{o}^1, \hat{o}^2, ..., \hat{o}^n\}$)
    Дано: множество квантизованных траекторий $\hat{O}$
    Найти: контейнер типа Ключ-Значение,
    реализующий отображение $K\left(\hat{c}\right)$
    Создаем пустой контейнер типа Ключ-Значение
    $K \leftarrow []$
    пробегаем по всем траекториям
    for i=1,. . . ,n **do**
        пробегаем по точкам траектории
        for j=1,. . . ,m **do**
            добавляем номер объекта o^i во все ячейки из $D\left(\hat{c}_j^i\right)$
            for dx=-2,0,2 **do**
                for dy=-2,0,2 **do**
                    вычисляем обобщенный индекс соседней ячейки
                    $\hat{c}_0 \leftarrow w\left(\hat{x}\left(\hat{c}_j^i\right) + dx, \hat{y}\left(\hat{c}_j^i\right) + dy, \hat{t}\left(\hat{c}_j^i\right) + dt\right)$
                    добавляем o_i в множество объектов ячейки $\hat{c}_0$
                    $K\left[\hat{c}_0\right].insert(i)$
                end for
            end for
        end for
    end for
    return res;
end function

$$\hat{o}^j = \left(\hat{c}_1^j, \ldots, \hat{c}_m^j\right).$$

Будем называть пару траекторий $\left(o^i, o^j\right)$ *похожими*, если существуют номера
$a, b \in \{1, ..., m\}$, $a < b$, такие что

$$t\left(\hat{c}_b^i\right) - t\left(\hat{c}_a^i\right) \geq \left(t_{\min} - \Delta_t\right),$$

и для любого $k \in \{a, ..., b\}$ имеет место $\hat{c}_k^i \in D\left(\hat{c}_k^j\right)$, где $D\left(\hat{c}_k^j\right)$ - множество ячеек, соседних с $\hat{c}_k^j$.

Пусть $\left(o^i, o^j\right)$ - траектории-компаньоны. Следовательно, на всяком совместном участке $[t_u, t_v]$ в любой момент времени $t_r \in [t_u, t_v]$ имеет место:

$$d\left(p_r^i, p_r^j\right) \leq d_{\min}.$$

Ранее мы наложили ограничение на параметр квантизации:

$$\Delta_s = \frac{d_{\min}}{\sqrt{2}}.$$

Следовательно, $\hat{c}_r^i \in D\left(\hat{c}_r^i\right)$.

А так как продолжительность совместного участка у траекторий-компаньонов равна не менее, чем t_{min}, то траектории $\left(o^i, o^j\right)$ также будут являться похожими. Следовательно, всякие траектории-компаньоны являются похожими. Обратное в общем случае не верно.

Обозначим $S\left(\hat{o}\right)$ — множество траекторий, похожих на $\hat{o}$.

Описанный ранее клеточный индекс позволяет построить эффективный алгоритм, который для заданной траектории находит множество всех похожих на нее траекторий.

На верхнем уровне данный алгоритм можно описать так: пробегаем скользящим окном (отрезком) $[a, b]$, таким что $\left(t\left(\hat{c}_b\right) - t\left(\hat{c}_a\right)\right) \geq t_{\min}$, по ячейкам квантизованной траектории $\hat{o} = \left(\hat{c}_1, \ldots, \hat{c}_m\right)$. Для каждого положения отрезка $[a, b]$ с помощью клеточного индекса находим все объекты, которые присутствуют во всех ячейках $\hat{c}_a, ..., \hat{c}_b$ и добавляем полученное множество в общий результат.

Заметим, что так как при построении клеточного индекса мы добавляли каждый объект не только в «свою» ячейку, но и во все соседние с ней, то все траектории, похожие на $\hat{o}$, будут включены в ответ.

Ниже приведено описание этого алгоритма (6.1).

Введем некоторые ограничения на множество траекторий и с их учетом дадим оценки сложности алгоритма поиска похожих траекторий.

Пусть $P, L \in \mathbb{R}$, а $p_0 = (x_0, y_0, t_0)$ — некоторая точка в пространстве-времени. Обозначим $U(p_0, P, L)$ — окрестность точки p_0:

Algorithm 6.1 Поиск траекторий, похожих на заданную

function GetSimilarTrajectories($\hat{o}, K$)
  Дано: квантизованная траектория $\hat{o} = (\hat{c}_1, \ldots, \hat{c}_m)$,
  клеточный индекс K
  Найти: S — множество траекторий, похожих на $\hat{o}$
  Создаем пустое множество траекторий
  $S \leftarrow []$, $a \leftarrow 1$, $t_{sum} \leftarrow 0$
  указатель b (правый край отрезка) пробегает по всем точкам
  траектории $\hat{o}$
  for $b = 1, \ldots, m$ **do**
    $t_{sum} \leftarrow t_{sum} + (t(\hat{c}_b) - t(\hat{c}_{b-1}))$
    нашли совместный участок достаточной продолжительности
    if $t_{sum} \geq t_{\min} - \Delta_t$ **then**
      пробуем переместить указатель a вперед
      $flag \leftarrow 1$
      while $flag \& (t(\hat{c}_a) < t(\hat{c}_b))$ **do**
        $flag \leftarrow 0$
        if $t_{sum} - (t(\hat{c}_{a+1}) - t(\hat{c}_a)) \geq t_{\min} - \Delta_t$ **then**
          $t_{sum} \leftarrow t_{sum} - (t(\hat{c}_{a+1}) - t(\hat{c}_a))$
          $a \leftarrow a + 1$
          $flag \leftarrow 1$
        end if
      end while
      if $a = b$ **then**
        break
      end if
      добавляем в S множество всех объектов,
      которые побывали во всех ячейках на отрезке $[a, b]$
      $S \leftarrow S \cup \bigcap_{j=a}^{b} K(\hat{c}_j)$
    end if
  end for
  return S
end function

$$U\left(p_0, P, L\right) = \left\{p = (x, y, t) \,||x - x_0| \leq P, |y - y_0| \leq P, |t - t_0| \leq L\right\}.$$

Пусть $D \in \mathbb{R}$, $C \in N$. Будем говорить, что множество траекторий $O = \{o^1, o^2, ..., o^n\}$ обладает (P, L, D, C)-свойством, если для всякой траектории $o^i = \left\langle p_1^i, ..., p_m^i \right\rangle$ и для любых $a, b \in \{1, ..., m\}$, таких что $\left(t_b^i - t_a^i\right) \geq D$ среди точек $p_a^i, ..., \hat{p}_b^i$ найдется такая точка p_j^i, что суммарное число траекторий других объектов из O, точки которых содержатся в $U\left(p_j^i, P, L\right)$, не превосходит величину C.

Содержательно (P, L, D, C)-свойство означает, что для всякой траектории из O на всяком отрезке достаточной длины найдется точка траектории, в (P, L)-окрестности которой общее число различных объектов не превышает заданный порог C.

Заметим, что реальные данные (траектории автомобилей, пешеходов и др.) обладают (P, L, D, C)-свойством при определенных значениях параметров P, L, D, C.

Рассмотрим последовательность множеств траекторий $O_1, ..., O_k, ...$, каждое из которых обладает (P, L, D, C)-свойством. Положим, что при стремлении k к бесконечности в множествах O_k число траекторий n и число точек в траекториях m также стремятся к бесконечности. Имеет место следующее утверждение.

Теорема 1. *Если для параметров квантизации Δ_s, Δ_t выполнено: $\Delta_s < P, \Delta_t < L$ и $t_{min} \geq D$, то при стремлении k к бесконечности время работы алгоритма 6.1, примененного к произвольной квантизованной траектории из множества $\hat{O}_k$, не превосходит величину $O\left(m^2\right)$ в среднем.*

Доказательство. Заметим, что сложность алгоритма 6.1 не превосходит величину $O(m) \cdot (A + B)$. Здесь первый множитель $O(m)$ соответствует числу возможных различных положений скользящего окна $[a, b]$, величина A — сложность вычисления операции пересечения множеств $\bigcap_{j=a}^{b} K\left(\hat{c}_j\right)$, а величина B — сложность вычисления операции объединения.

Из наличия (P, L, D, C)-свойства у множества траекторий O_k следует, что для всякой траектории o^i и положения скользящего окна $[a, b]$ среди точек траектории $p_a^i, ..., p_b^i$ найдется точка p_j^i, в окрестности которой — $U\left(p_j^i, P, L\right)$ общее число различных объектов не превышает C. Из определения операции квантизации следует, что точка p_j^i будет преобразована в некоторую ячейку $\hat{c}_j^i$ с размерами Δ_s, Δ_t. При этом сама точка p_j^i будет содержаться в ячейке $\hat{c}_j^i$. Так как $\Delta_s < P, \Delta_t < L$, следовательно ячейка $\hat{c}_j^i$ целиком содержится в окрестности $U\left(p_j^i, P, L\right)$. А значит

и общее число различных объектов, содержащихся в ячейке $\hat{c}_j^i$ не превосходит величину C. Далее, будем полагать, что клеточный индекс K представляет собой хеш-таблицу, элементы которой также представляют собой хеш-таблицы. Из этого следует, что $A = C \cdot (b-a) \cdot O(1) = O(m)$ в среднем. Здесь мы предполагаем что $b - a < m$ и для вычисления пересечения множеств достаточно проверить содержание элементов из множества $K\left(\hat{c}_j^i\right)$ во всех остальных множествах.

Сложность B, как легко видеть, линейным образом зависит от числа объектов в пересечении $\bigcap_{j=a}^{b} K\left(\hat{c}_j\right)$. Следовательно $B = O(m)$ в среднем.

В итоге получаем утверждение теоремы.

Перейдем к построению локального алгоритма поиска траекторий-компаньонов.

7. Локальный алгоритм поиска компаньонов

Ниже приведено описание локального алгоритма поиска траекторий-компаньонов. (7.1).

Необходимо отметить, что данный алгоритм осуществляет поиск точного решения задачи о компаньонах. Для каждой траектории сначала с помощью алгоритма 6.1 ищется множество всех похожих на нее траекторий, а затем с помощью последующей проверки предиката компаньонства 3.1 из найденного множества исключаются все траектории, которые не являются компаньонами.

Для последовательности множеств траекторий $O_1, ..., O_k, ...$ имеет место следующее утверждение.

Теорема 2. *Если для параметров квантизации Δ_s, Δ_t выполнено: $\Delta_s < P, \Delta_t < L$ и $t_{min} \geq D$, то при стремлении k к бесконечности время работы алгоритма 7.1, примененного к множеству O_k, обладающему (P, L, D, C)-свойством, не превосходит величину $O\left(n \cdot m^2\right)$ в среднем.*

Здесь, как и ранее, n — число траекторий в O_k, m — число точек в каждой траектории. При стремлении k к бесконечности параметры n и m также стремятся к бесконечности.

Доказательство. Из описания алгоритма 7.1 видно, что его временная сложность равна $O\left(n \cdot (V + W \cdot F)\right)$, где V — сложность работы алгоритма 6.1, а W — максимальное число траекторий, похожих на заданную, F — сложность вычисления предиката компаньонства.

Из теоремы [?] следует, что $V = O\left(m^2\right)$ в среднем.

Далее заметим, что из наличия у множества траекторий (P, L, D, C)-свойства следует, что общее число траекторий, похожих на заданную, не

Algorithm 7.1 Локальный алгоритм поиска траекторий-компаньонов

```
1: function LocalSearch( O )
2:     Дано: множество траекторий O
3:     Найти: W — множество пар-компаньонов
4:     строим множество квантизованных траекторий
5:     Ô ← Q (O)
6:     строим клеточный индекс
7:     K = BuildIndex(Ô)
8:     создаем пустое множество для сохранения результата
9:     W ← []
10:    пробегаем по всем траекториям
11:    for i=1,...,n do
12:        ищем траектории, похожие на ô^i
13:        S^i ← GetSimilarTrajectories(ô^i, K)
14:        for k = 1, ..., |S^i| do
15:            проверяем, являются ли траектории компаньонами
16:            g = TestPair (o^i, o^k)
17:            if g=1 then
18:                помещаем пару траекторий в результат
19:                W.insert (o^i, o^k)
20:            end if
21:        end for
22:    end for
23:    return W
24: end function
```

может быть более, чем $C \cdot O(m)$, так как для всякого положения скользящего окна $[a, b]$ найдется ячейка, в которой содержится не более C объектов (см. алгоритм 6.1). Следовательно, $W = C \cdot O(m)$. Ранее (3) было показано, что $F = O(m)$.

Следовательно, сложность локального алгоритма поиска равна $O\left(m^2 \cdot n\right)$ в среднем. Теорема доказана.

Таким образом, алгоритм 7.1 имеет линейную сложность относительно числа объектов, что делает его ценным с практической точки зрения, так как обычно на практике число объектов n значительно превышает число точек в траектории — m.

8. Распределенный алгоритм поиска

В данном разделе описан распределенный алгоритм поиска пар траекторий-компаньонов. Описание алгоритма использует стандартную терминологию (RDD, map, reduce, экзекьютор и др.), широко применяемую в распределенных вычислительных системах [4], [5], [6].

Ранее, в разделе 4, была введена функция квантизации $\hat{c}$, которая ставит в соответствие каждой точке пространства-времени p целочисленный индекс ячейки. При этом, все ячейки имеют одинаковые размеры: Δ_s — по широте и долготе и Δ_t — по времени.

Можно считать, что функция $\hat{c}$ задается параметрами Δ_s и Δ_t. То есть $\hat{c}\,(p) = \hat{c}\,(\Delta_s, \Delta_t, p)$.

Рассмотрим функцию $\hat{Z}\,(p) = \hat{c}\,(D_s, D_t, p)$, где параметры D_s и D_t принимают значения, значительно большие, чем Δ_s и Δ_t. Функция $\hat{Z}$ разбивает пространство-время на макро-ячейки значительно большего размера. Данные макро-ячейки будем называть *блоками*.

Распределенный алгоритм основан на разбиении пространства-времени на блоки. Размеры блоков D_s и D_t являются конфигурируемыми и должны подбираться так, чтобы общий объем точек, попадающих в блок, был достаточно мал, чтобы полностью поместиться в оперативную память одного экзекьютора.

Распределенный алгоритм состоит из трех этапов:

1) разбиение исходных данных по блокам;

2) параллельная работа локальных алгоритмов поиска (7.1) на блоках;

3) сбор результатов работы локальных алгоритмов на центральной машине кластера.

Заметим, что при разбиении пространства-времени на блоки некоторые траектории-компаньоны могут оказаться потерянными. Это может

произойти, если, например, части совместного участка компаньонства располагаются в соседних блоках. При этом продолжительность каждой из этих частей не превышает параметр t_{min}. Данный недостаток может быть легко устранен за счет использования перекрывающихся блоков. При этом размер перекрытия по времени должен быть не менее, чем t_{min}, а размер перекрытия по каждой из координат должен быть не менее, чем $t_{min} \cdot V_{max}$, где V_{max} - максимально возможная скорость движения объекта. Далее, для простоты мы будем полагать, что блоки не перекрываются.

Отметим, что при фиксированном числе экзекьюторов временная сложность распределенного алгоритма асимптотически равна сложности локального алгоритма — $O\left(n \cdot m^2\right)$, где число траекторий — n и число точек в траекториях — m одновременно стремятся к бесконечности (см. раздел 7.1).

Ниже приведено детальное описание распределенного алгоритма (8.1).

Algorithm 8.1 Распределенный алгоритм поиска траекторий компаньонов

function DISTRIBUTEDSEARCH(O)
    Дано: распределенное множество точек траекторий $O = RDD[p]$
    Найти: W — множество пар траекторий-компаньонов
    Каждой точке p ставим в соответствие пару $\left(\hat{Z}\left(p\right), p\right)$,
    где $\hat{Z}\left(p\right)$ - номер блока b, в котором находится точка p.
    $RDD[b, p]\, G \leftarrow O.map\left(p \Rightarrow \left(\hat{Z}\left(p\right), p\right)\right)$
    Группируем точки по номерам блоков
    $RDD[b, O_b]\, B \leftarrow G.groupByKey$
    Здесь B представляет собой RDD-контейнер типа
    Ключ-Значение, где ключом выступает номер блока b,
    а значением — множество точек, которые принадлежат
    блоку — O_b.
    Заметим, что после операции groupByKey каждое множество O_b
    будет гарантированно размещено целиком на одном экзекьюторе.
    Далее, для каждого блока запускаем локальный алгоритм (7.1)
    $RDD[\left(o^i, o^j\right)]\, H \leftarrow B.map\left((b, O_b) \Rightarrow LocalSearch\left(O_b\right)\right)$
    Устраняем возможные повторения и собираем результат
    на управляющей машине кластера
    $W \leftarrow H.distinct.collect$
    return W
end function

Список литературы

[1] K. Zheng, Y. Zheng, N. Yuan, S. Shang and X. Zhou, “Online Discovery of Gathering Patterns over Trajectories”, *IEEE Transactions on Knowledge and Data Engineering*, **26**:8 (2014), 1974–1988.

[2] L. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, C. Hung and W. Peng, “On Discovery of Traveling Companions from Streaming Trajectories”, IEEE 28th International Conference on Data Engineering, 2012, 186–197.

[3] Z. Li, B. Ding, J. Han and R. Kays, “Swarm: Mining Relaxed Temporal Moving Object Clusters”, *PVLDB*, **3**:1 (2010), 723–734.

[4] “Programming model "map-reduce"”, https://en.wikipedia.org/wiki/MapReduce.

[5] “Apache Spark Framework”, https://spark.apache.org/.

[6] B.Chambers, M.Zaharia, *Spark: The Definitive Guide: Big Data Processing Made Simple*, 2018.

[7] “Equirectangular Projection”, https://en.wikipedia.org/wiki/Equirectangular_projection.

References

[1] K. Zheng, Y. Zheng, N. Yuan, S. Shang and X. Zhou, “Online Discovery of Gathering Patterns over Trajectories”, *IEEE Transactions on Knowledge and Data Engineering*, **26**:8 (2014), 1974–1988.

[2] L. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, C. Hung and W. Peng, “On Discovery of Traveling Companions from Streaming Trajectories”, IEEE 28th International Conference on Data Engineering, 2012, 186–197.

[3] Z. Li, B. Ding, J. Han and R. Kays, “Swarm: Mining Relaxed Temporal Moving Object Clusters”, *PVLDB*, **3**:1 (2010), 723–734.

[4] “Programming model "map-reduce"”, https://en.wikipedia.org/wiki/MapReduce.

[5] “Apache Spark Framework”, https://spark.apache.org/.

[6] B.Chambers, M.Zaharia, *Spark: The Definitive Guide: Big Data Processing Made Simple*, 2018.

[7] “Equirectangular Projection”, https://en.wikipedia.org/wiki/Equirectangular_projection.

Distributed companion trajectory search algorithm

Sokolov A.P., Aliseychik P.A., Moiseev S.V.

The work is focused on development of the distributed algorithm for search of companion-trajectories. We call two trajectories as companion-trajectories if they both contain section of required duration on which at any time moment distance between two objects is no larger than specified threshold. It’s assumed that search is performed on database that is so big that it cannot be stored and processed on a single work station. Such tasks are usually called big-data tasks. We define trajectory quantization procedure, introduce cell index data structure, define similar trajectories. Then we build effective algorithm for search of similar trajectories. Based on this algorithm we develop effective

local and distributed algorithms for search of companion trajectories. Asymptotic complexity of these algorithms is estimated.

Keywords: big-data, geo-spatial data analysis, trajectory analysis, companion-trajectory search, similar trajectory search, traffic analysis.