Алгоритмы и методы решения задач составления расписаний и других экстремальных задач на графах больших размерностей

Е. В. ПАНКРАТЬЕВ
Московский государственный университет им. М. В. Ломоносова

А. М. ЧЕПОВСКИЙ
Московский государственный технический университет им. Н. Э. Баумана

Е. А. ЧЕРЕПАНОВ, С. В. ЧЕРНЫШЁВ
Московский государственный университет им. М. В. Ломоносова

УДК 519.68

Ключевые слова: нахождение кратчайших путей в графе, эволюционные методы.

Аннотация

Рассматривается ориентированный граф $G = (V, E)$ большой размерности, на рёбрах которого задан набор характеристик. В графе выделено подмножество вершин $V' \subset V$, на которые наложены дополнительные условия. В работе приведён алгоритм сведения задачи оптимизации на графе G к задаче оптимизации на графе $G' = (V', E')$ меньшей размерности. Приведены основные этапы решения и методы построения приближённого решения задачи на преобразованном графе G'.

Abstract

E. V. Pankratiev, A. M. Chepovskii, E. A. Cherepanov, S. V. Chernyshev, Algorithms and methods for solving scheduling problems and other extremum problems on large-scale graphs, Fundamentalnaya i prikladnaya matematika, vol. 9 (2003), no. 1, pp. 235—251.

We consider a large-scale directed graph $G = (V, E)$ whose edges are endowed with a family of characteristics. A subset of vertices of the graph, $V' \subset V$, is selected and some additional conditions are imposed on these vertices. An algorithm for reducing the optimization problem on the graph G to an optimization problem on the graph $G' = (V', E')$ of a lower dimension is developed. The main steps of the solution and some methods for constructing an approximate solution to the problem on the transformed graph G' are presented.

Введение

В работе рассматриваются оптимизационные задачи на графах большой размерности, не имеющие эффективных алгоритмов нахождения точного решения.

Фундаментальная и прикладная математика, 2003, том 9, № 1, с. 235—251.

В качестве примера можно рассмотреть задачу построения оптимальных маршрутов движения парка автомобилей. Задана сетка дорог с большим количеством узлов: перекрёстков, тупиков и точек обслуживания, через которые должны пройти маршруты движения транспорта. Сетке дорог ставится в соответствие ориентированный граф, вершинами которого являются узлы данной сетки, а рёбрами — отрезки дорог между узлами (дороги могут быть односторонними). Каждому ребру приписывается длина — расстояние между соответствующими узлами сетки. Точки обслуживания трактуются как отмеченные вершины с весами, характеризующими данную точку. Весом может быть, например, количество контейнеров или объём груза, которые должна собрать машина в этой точке. Ищется набор оптимальных маршрутов, начинающихся и заканчивающихся в заданных точках (гаражах) и ограниченных суммарным количеством весов (например, количеством собираемых одной машиной контейнеров) и некоторой функцией от длин рёбер графа, которая может учитывать физическую длину маршрута (километраж), либо время движения транспорта, либо стоимостные характеристики маршрута движения.

Другой пример — задача составления оптимального расписания работы агентов, обслуживающих клиентов. Обслуживание заключается в доставке (сборе) некоторых объектов или предоставлении услуг. Каждый объект может иметь несколько характеристик (например, массу, размер, стоимость и т. д.). Для каждого агента заданы начало и конец движения, интервал работы, ограничения на суммарные характеристики объектов. Для каждого клиента заданы интервал времени, в течение которого должна быть оказана услуга, время, затрачиваемое на оказание услуги, и характеристики объекта, который он хочет получить или отдать. Под расписанием будем понимать график работы каждого агента: выбор маршрутов движения и времени обслуживания. Графики должны быть составлены так, чтобы были обслужены все клиенты и выполнены ограничения по времени и характеристикам объектов. Расписание необходимо составить таким образом, чтобы минимизировать общие затраты агентов (например, время работы, суммарные транспортные расходы, количество задействованных агентов и др.).

В первом разделе работы описывается класс решаемых задач и их математические постановки. Приводится метод декомпозиции исходной задачи на более простые независимые подзадачи, алгоритмы решения которых приводятся во втором и третьем разделах. В заключении рассматриваются достоинства и недостатки методов, изложенных в данной работе.

1. Общий метод решения оптимизационных задач

1.1. Класс рассматриваемых задач

В работе рассматриваются задачи, которые характеризуются следующими условиями:

1) задан ориентированный граф $G = (V, E)$ большой размерности, $|V| \sim 10^6$;
2) на рёбрах графа задан набор характеристик;
3) в графе G выделено подмножество вершин $V' \subset V$, $|V'| \sim 10^3$;
4) заданы характеристики вершин V';
5) задана целевая функция F;
6) необходимо найти набор маршрутов (путей в графе G), минимизирующих значение целевой функции;
7) заданы ограничения на построение набора маршрутов в вершинах V';
8) заданы ограничения на отдельные маршруты в виде функций характеристик рёбер и выделенных вершин, входящих в данный маршрут.

1.2. Математическая постановка

Методы, приведённые в данной работе, в какой-то мере применимы ко всему классу перечисленных оптимизационных задач. Чтобы не усложнять выкладки, всюду в дальнейшем будем рассматривать только задачу составления оптимального расписания работы агентов.

Дан ориентированный граф $G = (V, E)$, на рёбрах которого определены две неотрицательные весовые функции: длина и время движения по ребру.

Есть выделенное множество вершин $C = \{c_1, \ldots, c_n\} \subset V$ — местоположения клиентов. Для каждого клиента (следовательно, для каждой вершины c_i, $i = \overline{1,n}$) задан набор неотрицательных чисел — характеристик объекта $\mathbf{c}_i^{\gamma} = (\gamma_i^1, \gamma_i^2, \ldots, \gamma_i^q)$, интервал времени работы $\mathbf{c}_i^{\tau} = (\tau_i^1, \tau_i^2)$ и время, необходимое для оказания услуги $\hat{\tau}_i$.

Каждый агент начинает свою работу в некоторой заданной наперёд точке s_j и заканчивает работу в точке f_j; $s_j, f_j \in V$, $j = \overline{1,m}$. Множество всех этих точек обозначим $A = \{s_1, f_1, \ldots, s_m, f_m\} \subset V$. Для агента задан набор неотрицательных чисел — ограничение на суммарные характеристики объектов $\mathbf{a}_i^{\alpha} = (\alpha_i^1, \alpha_i^2, \ldots, \alpha_i^q)$ и интервал времени работы $\mathbf{a}_i^{\tau} = (\tau_i^1, \tau_i^2)$.

Определение 1.1. *Характеристикой ребра* $e \in E$ назовём вектор $\boldsymbol{e}$, $\boldsymbol{e} \in \mathbb{R} \times \mathbb{R}$, $\boldsymbol{e} = (e^d, e^t)$, где e^d, e^t — длина и время движения.

Определение 1.2. *Маршрутом j-го агента* $\mathbf{r}_j$ назовём последовательность вершин, через которые проезжает агент, последовательность соединяющих их рёбер, время прибытия агента в каждую вершину и множество клиентов, которых он обслужил (агент может проехать через выделенную вершину, но не обслуживать её). Таким образом, маршрут $\mathbf{r}_j$ задаётся следующими наборами:

1) $(v_j^0, \ldots, v_j^{l_j})$, где $v_j^0 = s_j$, $v_j^{l_j} = f_j$ и $v_j^i \in V$, $i = \overline{0, l_j}$;
2) $(e_j^1, \ldots, e_j^{l_j})$, где $e_j^i = (v_j^{i-1}, v_j^i)$ и $e_j^i \in E$, $i = \overline{1, l_j}$;
3) $(t_j^0, \ldots, t_j^{l_j})$, где t_j^i означает время прибытия агента в вершину v_j^i, $i = \overline{0, l_j}$;

4) $C_j = \{c_j^{k_1}, \ldots, c_j^{k_{n_j}}\} \subset C$, где k_i — номер вершины маршрута, в которой происходит обслуживание клиента, т. е. $c_j^{k_i} = v_j^{k_i}$.

Определение 1.3. *Расписанием* $\mathbf{R}$ назовём множество маршрутов движения агентов $\{\mathbf{r}_1, \ldots, \mathbf{r}_m\}$.

Определение 1.4. Маршрут $\mathbf{r}_j$ называется *корректным*, если выполнены следующие ограничения:

1) суммарные по характеристикам:

$$\sum_{i=1}^{n_j} (\mathbf{c}^\gamma)_j^{k_i} \leqslant \mathbf{a}_j^\alpha;$$

2) по интервалу времени работы агента:

$$t_j^i \in \mathbf{a}_j^\tau, \quad i = \overline{0, l_j};$$

3) по интервалу времени работы клиента:

$$t_j^{k_i} \in (\mathbf{c}^\tau)_j^{k_i}, \quad i = \overline{1, n_j};$$

4) по времени движения по рёбрам графа:

для всех $1 \leqslant i \leqslant l_j$ $\begin{cases} (e^t)_j^i \leqslant t_j^i - t_j^{i-1} - \hat{\tau}_j^{i-1}, & \text{если } (i-1) \in \{k_1, \ldots, k_{n_j}\}, \\ (e^t)_j^i \leqslant t_j^i - t_j^{i-1} & \text{в противном случае.} \end{cases}$

Определение 1.5. Расписание $\mathbf{R}$ назовём *корректным*, если

1) все маршруты движения агентов корректны;

2) $C = C_1 \cup \ldots \cup C_m$ (обслужены все клиенты);

3) $C_i \cap C_j = \varnothing$, $i \neq j$ (никакие два агента не обслуживают одного клиента).

Среди всех корректных расписаний необходимо выбрать оптимальное. Для этой цели введём некоторую целевую функцию $F(\mathbf{R})$. Будем считать, что расписание $\mathbf{R}_1$ более оптимально, чем расписание $\mathbf{R}_2$, если $F(\mathbf{R}_1) < F(\mathbf{R}_2)$. Тем самым задача оптимизации сводится к поиску такого корректного расписания, которое минимизирует целевую функцию.

Целевая функция выбирается из дополнительных соображений. Обычно при решении оптимизационных задач на составление расписания мы имеем ограничение по времени и пытаемся уменьшить суммарные транспортные расходы. Возможно, есть и другие требования к расписанию, например равномерная загруженность агентов. Поэтому обычно целевая функция зависит от характеристик рёбер, от суммарных характеристик объектов, входящих в маршруты расписания, и т. д.

1.3. Разбиение оптимизационных задач на две независимые подзадачи

В нашей задаче граф G имеет большую размерность, а количество клиентов и агентов относительно мало. Поэтому вместо исходного графа $G = (V, E)$ мы будем работать с меньшим графом $G' = (V', E')$, где $V' = A \cup C$, а множество E' — всевозможные кратчайшие пути в графе G между вершинами из V'. Таким образом, исходную задачу можно разбить на две подзадачи:

1) построение по исходному графу G преобразованного графа G';

2) решение оптимизационной задачи на графе G'.

Для первой подзадачи существуют эффективные методы решения. В случае, если вектор характеристики ребра одномерный, для поиска кратчайших путей в графе применимы алгоритмы Дейкстры, Флойда—Уоршолла, Джонсона и др. В настоящее время для случая неориентированного графа известны модификации алгоритма Дейкстры, работающие за линейное время от количества рёбер (см. [2]).

Однако не всегда можно ввести понятие кратчайшего пути (например, в задаче составления оптимального расписания работы агентов один путь может быть оптимальным по времени, а другой — по длине). Соответственно, возникает необходимость в поиске оптимальных путей в графе, характеристики рёбер которого имеют размерность 2 и больше. В этом случае все перечисленные алгоритмы становятся практически непригодными для решения поставленной задачи.

В следующем разделе приводится алгоритм нахождения кратчайшего пути по первой компоненте характеристики ребра от заданной вершины до всех вершин графа при наложенных ограничениях на остальные компоненты.

Отметим, что первая подзадача предполагает построение полного графа G'. Однако жизненная подоплёка задачи оптимизации такова, что нет необходимости искать оптимальные пути между всеми парами вершин из V', а достаточно найти оптимальные пути между относительно «близко» расположенными вершинами. Критерий отсечения можно формировать, исходя из теоретических соображений и условия поставленной задачи или же анализируя полученные практические результаты. Например, можно ввести ограничения на степени вершин графа G'.

К сожалению, для решения второй подзадачи в настоящее время эффективных алгоритмов не существует. Кроме того, нет и общей теории построения алгоритмов нахождения приближённых решений. Поэтому для каждой конкретной задачи приходится искать свой способ решения.

Общая идеология и возможные методы решения второй подзадачи приводятся в третьем разделе.

2. Алгоритм построения кратчайших путей в ориентированных графах с весовой вектор-функцией рёбер

2.1. Постановка задачи

Имеется ориентированный граф $G = (V, E)$, на рёбрах которого определены две неотрицательные весовые функции: длина и время движения по ребру.

В графе выделена одна из вершин $s \in V$ — начало движения. Задача состоит в том, чтобы для каждой вершины $v \in V$ и произвольного времени $t \in \mathbb{R}$ найти длину $\mathbf{D}(v, t)$ кратчайшего пути из вершины s в вершину v, время движения вдоль которого не превосходит t. Отметим, что функция $\mathbf{D}(v, t)$ определена не полностью: при некоторых значениях v и t может не существовать пути, удовлетворяющего условию. Это может произойти в том случае, если время t слишком мало или точки s и v вообще не связаны.

Заметим, что исходный граф G может содержать кратные рёбра и петли. Наличие петель не влияет на результат, однако от кратных рёбер избавиться невозможно.

2.2. Определения и обозначения

Определение 2.1. Множество рёбер E разобьём на непересекающиеся подмножества E_{vw} — множества рёбер с началом в вершине v и концом в вершине w.

Определение 2.2. *Путём* назовём последовательность рёбер $(e_1 \dots e_n)$, такую что начало ребра e_i является концом ребра e_{i-1}, $i = \overline{2, n}$.

Определение 2.3. Множество L всевозможных путей разобьём на непересекающиеся подмножества L_{vw} — множества путей с началом в точке v и концом в точке w. В нашей задаче все маршруты начинаются в вершине s.

Определение 2.4. *Характеристикой пути* $l = (e_1 \dots e_n)$ назовём вектор $\boldsymbol{l} \in \mathbb{R} \times \mathbb{R}$, $\boldsymbol{l} = (l^d, l^t)$, где $l^d = \sum\limits_{i=1}^{n} e_i^d$, $l^t = \sum\limits_{i=1}^{n} e_i^t$.

Определение 2.5. Обозначим $\mathbf{L} \subset \mathbb{R} \times \mathbb{R}$ множество характеристик всевозможных путей в графе. Будем считать характеристики $\boldsymbol{l}_1, \boldsymbol{l}_2 \in \mathbf{L}$ равными $(\boldsymbol{l}_1 = \boldsymbol{l}_2)$, если и только если $l_1^d = l_2^d$ и $l_1^t = l_2^t$.

Введём отношение частичного порядка на множестве $\mathbf{L}$. Будем считать, что $\boldsymbol{l}_1 < \boldsymbol{l}_2$, если $l_1^d \leqslant l_2^d$ и $l_1^t \leqslant l_2^t$, причём одно из неравенств строгое. Также введём отношение линейного порядка $\prec$: будем считать, что $\boldsymbol{l}_1 \prec \boldsymbol{l}_2$, $\boldsymbol{l}_1, \boldsymbol{l}_2 \in \mathbf{L}$, если $l_1^t < l_2^t$ или $l_1^t = l_2^t$ и $l_1^d < l_2^d$.

Определим операцию сложения двух характеристик $\boldsymbol{l}_1, \boldsymbol{l}_2 \in \mathbf{L}$. Суммой $\boldsymbol{l}_1 + \boldsymbol{l}_2$ назовём вектор $\boldsymbol{l} = (l_1^d + l_2^d, l_1^t + l_2^t)$. Отметим, что в общем случае $\boldsymbol{l}$ не обязан принадлежать множеству $\mathbf{L}$.

Определение 2.6. Путь $l \in L_{sv}$ назовём *оптимальным*, если не существует пути $\hat{l} \in L_{sv}$, такого что $\hat{\boldsymbol{l}} < \boldsymbol{l}$.

Определение 2.7. Множество $\mathbf{Q} \subset \mathbf{L}$ назовём *несоизмеримым*, если любые две характеристики, принадлежащие этому множеству, несравнимы относительно отношения частичного порядка $<$.

Определение 2.8. *Минимальным подмножеством* множества $\mathbf{Q} \subset \mathbf{L}$ назовём такое несоизмеримое подмножество $\mathrm{MIN}(\mathbf{Q})$ множества $\mathbf{Q}$, что для любой характеристики $\boldsymbol{q} \in \mathbf{Q}$ найдётся характеристика $\hat{\boldsymbol{q}} \in \mathrm{MIN}(\mathbf{Q})$, такая что $\hat{\boldsymbol{q}} \leqslant \boldsymbol{q}$.

Докажем что определение минимального множества корректно, т. е. не существует двух минимальных подмножеств. Предположим, что $\mathbf{Q}_1$ и $\mathbf{Q}_2$ — различные минимальные подмножества множества $\mathbf{Q} \subset \mathbf{L}$. Рассмотрим произвольный элемент $\boldsymbol{q} \in \mathbf{Q}_1 \setminus \mathbf{Q}_2$. Так как $\mathbf{Q}_2$ минимально, то существует элемент $\boldsymbol{q}_2 \in \mathbf{Q}_2$, такой что $\boldsymbol{q}_2 \leqslant \boldsymbol{q}$, но $\mathbf{Q}_1$ — тоже минимальное подмножество, значит, существует $\boldsymbol{q}_1 \in \mathbf{Q}_1$, такой что $\boldsymbol{q}_1 \leqslant \boldsymbol{q}_2$, поэтому $\boldsymbol{q}_1 \leqslant \boldsymbol{q}_2 \leqslant \boldsymbol{q}$. Но множество $\mathbf{Q}_1$ несоизмеримо, значит, $\boldsymbol{q} = \boldsymbol{q}_1 \implies \boldsymbol{q}_2 = \boldsymbol{q}$, но $\boldsymbol{q} \notin \mathbf{Q}_2$. Получили противоречие, поэтому $\mathbf{Q}_1 \setminus \mathbf{Q}_2 = \varnothing$. Аналогично доказывается, что $\mathbf{Q}_2 \setminus \mathbf{Q}_1 = \varnothing$, значит, $\mathbf{Q}_1 = \mathbf{Q}_2$.

2.3. Переформулировка задачи

Несложно показать, что функция $\mathbf{D}\colon V \times \mathbb{R} \to \mathbb{R}$ является монотонно невозрастающей кусочно-постоянной по второму параметру. Тем самым задачу можно сформулировать в таком виде: для каждой вершины $v \in V$ необходимо найти множество характеристик оптимальных путей $\mathbf{P}_v \subset \mathbf{L}$, $\mathbf{P}_v = \{\boldsymbol{p}_i \mid i = \overline{1, n_v}\}$, таких что $\mathbf{D}(v, t) = \boldsymbol{p}_j^d$, где $j = \max\limits_{\boldsymbol{p}_i^t \leqslant t} i$, причём $\boldsymbol{p}_i \prec \boldsymbol{p}_k$, если $i < k$.

2.4. Предварительные замечания

Предложенный здесь алгоритм использует ту же идею, что и алгоритм Дейкстры для нахождения кратчайших путей, и заключается в том, что мы поочерёдно перебираем вершины графа в порядке возрастания параметра t и оптимизируем найденные ранее пути.

Для каждой вершины v мы храним конечные множества характеристик $\mathbf{P}_v, \mathbf{Q}_v \subset \mathbf{L}$. В начале эти множества полагаем пустыми, а в процессе работы алгоритма будем добавлять характеристики найденных путей во множество $\mathbf{Q}_v$, удалять характеристики из $\mathbf{Q}_v$, если они не удовлетворяют некоторым условиям, и перемещать характеристики из $\mathbf{Q}_v$ в $\mathbf{P}_v$, если они удовлетворяют нашим условиям. Будем следить за тем, чтобы множество $\mathbf{Q}_v$ было несоизмеримым, а множество $\mathbf{P}_v$ будет несоизмеримым автоматически.

Кроме того, считаем, что у нас есть множество выделенных вершин $S \in V$ — это те вершины, для которых множества $\mathbf{Q}_v$ непусты. В процессе работы множества $\mathbf{Q}_v$, $\mathbf{P}_v$ и S будут изменяться.

В результате работы алгоритма мы получаем множества $\mathbf{P}_v$ — характеристики оптимальных путей.

Определение 2.9. *Весом* $\boldsymbol{w}(v)$ вершины $v \in S$ назовём характеристику $\boldsymbol{q} \in \mathbf{Q}_v$, такую что для любой характеристики $\hat{\boldsymbol{q}} \in \mathbf{Q}_v$ выполнено неравенство $\boldsymbol{q} \preceq \hat{\boldsymbol{q}}$.

Для работы алгоритма нам потребуются следующие функции:

- INSERT$(v, \boldsymbol{p})$. Аргументами этой функции являются вершина v и характеристика $\boldsymbol{p}$ найденного пути p до неё. Если существует характеристика $\hat{\boldsymbol{p}} \in \mathbf{P}_v$, такая что $\hat{\boldsymbol{p}} \leqslant \boldsymbol{p}$, то множества $\mathbf{Q}_v$ и S не изменяются, в противном случае
$$\mathbf{Q}_v^{\text{нов}} = \text{MIN}(\mathbf{Q}_v \cup \{\boldsymbol{p}\}), \quad S^{\text{нов}} = S \cup \{v\};$$
- EXTRACT$(v, \boldsymbol{p})$. Эта функция возвращает вершину v, являющуюся концом оптимального пути p, и его характеристику $\boldsymbol{p}$. Принцип её работы заключается в следующем:
  1) вершину $v \in S$ находим из условия $\boldsymbol{w}(v) \preceq \boldsymbol{w}(w)$ для любой $w \in S$,
  2) полагаем $\boldsymbol{p} = \boldsymbol{w}(v)$.

  В результате работы этой функции множества $\mathbf{Q}_v$, $\mathbf{P}_v$ и S изменяются следующим образом:
$$\begin{aligned} &\mathbf{Q}_v^{\text{нов}} = \mathbf{Q}_v \setminus \{\boldsymbol{p}\}; \\ &\mathbf{P}_v^{\text{нов}} = \mathbf{P}_v \cup \{\boldsymbol{p}\}; \end{aligned} \qquad S^{\text{нов}} = \begin{cases} S \setminus \{v\}, & \text{если } \mathbf{Q}_v^{\text{нов}} = \varnothing, \\ S & \text{в противном случае.} \end{cases}$$

2.5. Алгоритм

1. $S = \varnothing$
2. $\forall v \in V$ присвоить $\mathbf{Q}_v = \varnothing$, $\mathbf{P}_v = \varnothing$
3. INSERT$(s, \mathbf{0})$
4. Пока $S \neq \varnothing$
   4.1. EXTRACT$(v, \boldsymbol{p})$
   4.2. $\forall w \in V$ $\forall e \in E_{vw}$ INSERT$(w, \boldsymbol{p} + \boldsymbol{e})$

2.6. Корректность работы алгоритма

Процесс выполнения алгоритма будем рассматривать как последовательность переходов от одного состояния вычисления к другому. Будем считать, что выполнение одной строки алгоритма — неделимая операция. В нашем случае состояние характеризуется наборами множеств $\mathbf{P}_v$, $\mathbf{Q}_v$, множеством S, значением переменных v и $\boldsymbol{p}$, а также номером выполняемой строки алгоритма. Обозначим $\mathbb{S}^{(n,k)}$ состояние, в котором находится алгоритм после выполнения k раз строки с

номером n. Тогда процесс выполнения алгоритма будет выглядеть следующим образом:

$$\mathbb{S}^{(1,1)} \to \mathbb{S}^{(2,1)} \to \mathbb{S}^{(3,1)} \to \mathbb{S}^{(4,1)} \to \mathbb{S}^{(4.1,1)} \to \mathbb{S}^{(4.2,1)} \to$$
$$\to \mathbb{S}^{(4,2)} \to \ldots \to \mathbb{S}^{(4,k)} \to \mathbb{S}^{(4.1,k)} \to \mathbb{S}^{(4.2,k)} \ldots \to \mathbb{S}^{(4,N+1)},$$

где N — общее количество итераций цикла 4. Обозначим $\mathbf{P}_v^k$, $\mathbf{Q}_v^k$, S^k, v^k, $\boldsymbol{p}^k$ множества $\mathbf{P}_v$, $\mathbf{Q}_v$, S и переменные v, $\boldsymbol{p}$ в состоянии $\mathbb{S}^{(4.1,k)}$.

Лемма 2.1. *Все фигурирующие в алгоритме характеристики соответствуют некоторым путям, начинающимся в вершине s.*

Доказательство. Достаточно проверить, что функция INSERT получает характеристику, соответствующую некоторому пути, так как это единственный способ появления новых характеристик в алгоритме. Докажем это утверждение по индукции.

База индукции: характеристике $\mathbf{0} = (0, 0)$ соответствует путь, не содержащий рёбер, ведущий из вершины s в s.

Переход индукции: если $l = (e_1 \ldots e_n)$ — путь, соответствующий характеристике $\boldsymbol{l}$, то $\hat{l} = (e_1 \ldots e_n\ e)$ — путь, соответствующий характеристике $\hat{\boldsymbol{l}} = \boldsymbol{l} + \boldsymbol{e}$.

Лемма 2.2. *Для всех $k < N$ выполнено неравенство*

$$\boldsymbol{p}^k \preceq \boldsymbol{p}^{k+1},$$

причём равенство возможно только в том случае, если $v^k \neq v^{k+1}$.

Доказательство. Рассмотрим следующие случаи.

1. Характеристика $\boldsymbol{p}^{k+1} \in \bigcup\limits_{w \in V} \mathbf{Q}_w^k$. Согласно определению функции EXTRACT $\boldsymbol{p}^k \preceq \boldsymbol{l}$ для любой характеристики $\boldsymbol{l} \in \bigcup\limits_{w \in V} \mathbf{Q}_w^k$, поэтому $\boldsymbol{p}^k \preceq \boldsymbol{p}^{k+1}$. Предположим, что $v^k = v^{k+1}$, тогда $\boldsymbol{p}^{k+1} \in \mathbf{Q}_{v^k}^k$, но $\boldsymbol{p}^k \notin \mathbf{Q}_{v^k}^k$, поэтому $\boldsymbol{p}^{k+1} \neq \boldsymbol{p}^k$.

2. Характеристика $\boldsymbol{p}^{k+1} \notin \bigcup\limits_{w \in V} \mathbf{Q}_w^k$. Значит, $\boldsymbol{p}^{k+1} = \boldsymbol{p}^k + \boldsymbol{e}$, где $\boldsymbol{e}$ — характеристика некоторого ребра, поэтому $\boldsymbol{p}^k \preceq \boldsymbol{p}^{k+1}$ и $v^k \neq v^{k+1}$.

Лемма 2.3. *Пусть $\mathbf{P}_v^N = \{\boldsymbol{p}^{k_1}, \boldsymbol{p}^{k_2}, \ldots, \boldsymbol{p}^{k_n}\}$, где $k_1 < k_2 < \ldots < k_n \leqslant N$. Тогда $(p^{k_1})^d > (p^{k_2})^d > \ldots > (p^{k_n})^d$.*

Доказательство. Пусть утверждение леммы не выполнено. Тогда существует i, такое что $(p^{k_i})^d \leqslant (p^{k_{i+1}})^d$. Согласно предыдущей лемме, $(p^{k_i})^t \leqslant (p^{k_{i+1}})^t$, поэтому $\boldsymbol{p}^{k_i} \leqslant \boldsymbol{p}^{k_{i+1}}$. Рассмотрим такой шаг $k < k_{i+1}$ и такое ребро e, что $\boldsymbol{p}^{k_{i+1}} = \boldsymbol{p}^k + \boldsymbol{e}$.

Предположим, что $k_i \leqslant k$. Тогда $\mathbf{P}_v^{k_i} \subset \mathbf{P}_v^k$ и $\boldsymbol{p}^{k_i} \in \mathbf{P}_v^k$. В состоянии $\mathbb{S}^{(4.2,k)}$ функция INSERT$(v, \boldsymbol{p}^k + \boldsymbol{e})$ добавила во множество $\mathbf{Q}_v^k$ характеристику $\boldsymbol{p}^{k_{i+1}} = \boldsymbol{p}^k + \boldsymbol{e}$. Значит, не должно существовать характеристики $\boldsymbol{l} \in \mathbf{P}_v^k$, такой что $\boldsymbol{l} \leqslant \boldsymbol{p}^{k_{i+1}}$, но $\boldsymbol{p}^{k_i} \leqslant \boldsymbol{p}^{k_{i+1}}$. Следовательно, предположение неверно и $k < k_i < k_{i+1}$.

Рассмотрим состояние $\mathbb{S}^{(4.1,k_i)}$. Множество $\mathbf{Q}_v^{k_i}$ получается в результате работы функции EXTRACT из некоторого множества $\hat{\mathbf{Q}}_v^{k_i}$ следующим образом: $\mathbf{Q}_v^{k_i} = \hat{\mathbf{Q}}_v^{k_i} \setminus \{\boldsymbol{p}^{k_i}\}$. Так как $\boldsymbol{p}^{k_i} \leqslant \boldsymbol{p}^{k_{i+1}}$ и множество $\hat{\mathbf{Q}}_v^{k_i}$ несоизмеримо, то $\boldsymbol{p}^{k_{i+1}} \notin \hat{\mathbf{Q}}_v^{k_i}$. Однако в процессе работы алгоритма характеристика $\boldsymbol{p}^{k_{i+1}}$ была добавлена во множество $\mathbf{Q}_v$ на шаге k, а удалена только на шаге k_{i+1}. Поскольку $k < k_i < k_{i+1}$, то $\boldsymbol{p}^{k_{i+1}} \in \hat{\mathbf{Q}}_v^{k_i}$, что приводит нас к противоречию. Лемма доказана.

Следствие. *Для всех $k \leqslant N$ и всех $v \in V$ множество $\mathbf{P}_v^k$ несоизмеримо.*

Доказательство. Утверждение непосредственно следует из доказательства леммы 2.3.

Лемма 2.4. *Для любого пути l найдётся $k \leqslant N$, такое что пути p^k и l заканчиваются в одной и той же вершине, причём $\boldsymbol{p}^k \leqslant \boldsymbol{l}$.*

Доказательство. Пусть $l = (e_1 \dots e_n)$. Обозначим $l_i = (e_1 \dots e_i)$, $i = \overline{1,n}$. Обозначим l_0 пустой путь, заканчивающийся в вершине s. Пусть v_i — конечная вершина пути l_i, $i = \overline{0,n}$. Докажем по индукции, что для всех $i = \overline{0,n}$ существует $k_i \leqslant N$, такое что для характеристики пути $p^{k_i} \in L_{sv_i}$ верно $\boldsymbol{p}^{k_i} \leqslant \boldsymbol{l}_i$, $\boldsymbol{p}^{k_i} \in \mathbf{P}_v^i$.

База индукции: $i = 0$. Положим $k_0 = 1$. Очевидно, что $\boldsymbol{p}^1 \leqslant \boldsymbol{l}_0$, $\boldsymbol{p}^1 \in \mathbf{P}_s$.

Переход: пусть утверждение леммы верно для всех $i \leqslant j$. Докажем утверждение для $i = j + 1$. На k_j-й итерации вызывается функция $\mathrm{INSERT}(v_{j+1}, \boldsymbol{p}^{k_j} + \boldsymbol{e}_{j+1})$. Если существует характеристика $\hat{\boldsymbol{p}} \in \mathbf{P}_{v_{j+1}}$, такая что $\hat{\boldsymbol{p}} \leqslant \boldsymbol{p}^{k_j} + \boldsymbol{e}_{j+1}$, то выбираем k_{j+1}, такое что $\boldsymbol{p}^{k_{j+1}} \leqslant \hat{\boldsymbol{p}}$. Тогда

$$\boldsymbol{p}^{k_{j+1}} \leqslant \boldsymbol{p}^{k_j} + \boldsymbol{e}_{j+1} \leqslant \boldsymbol{l}_j + \boldsymbol{e}_{j+1} = \boldsymbol{l}_{j+1}.$$

Если такой характеристики $\hat{\boldsymbol{p}}$ не существует, то $\boldsymbol{p}^{k_j} + e_{j+1}$ будет добавлена во множество $\mathbf{Q}_{v_{j+1}}$. На некотором шаге k_{j+1} функция EXTRACT вернёт характеристику $\boldsymbol{p}^{k_{j+1}} \in \mathbf{Q}_{v_{j+1}}$, такую что

$$\boldsymbol{p}^{k_{j+1}} \leqslant \boldsymbol{p}^{k_j} + \boldsymbol{e}_{j+1} \leqslant \boldsymbol{l}_j + \boldsymbol{e}_{j+1} = \boldsymbol{l}_{j+1},$$

что и требовалось доказать.

Лемма 2.5. *Для всех $k \leqslant N$ путь p^k является оптимальным.*

Доказательство. Предположим, что p^k не оптимальный путь ($\boldsymbol{p}^k \in \mathbf{P}_v^N$). Тогда по определению существует путь $l \in L_{sv}$, такой что $\boldsymbol{l} < \boldsymbol{p}^k$. Согласно лемме 2.4 существует путь $p^n \in L_{sv}$ ($\boldsymbol{p}^n \in \mathbf{P}_v^n$), такой что $\boldsymbol{p}^n \leqslant \boldsymbol{l}$. Поэтому $\boldsymbol{p}^n \leqslant \boldsymbol{l} < \boldsymbol{p}^k$. Но согласно следствию леммы 2.3 множество $\mathbf{P}_v^n$ несоизмеримо. Противоречие.

Теорема 2.1. *Алгоритм решает поставленную задачу за конечное время.*

Доказательство. Согласно леммам 2.2 и 2.4 все найденные пути оптимальны и различны. Так как в оптимальном пути нет повторяющихся рёбер, то таких путей конечное число, а значит, алгоритм закончит работу за конечное

время. Поскольку (лемма 2.3) для любого оптимального пути l алгоритм находит путь p, такой что $\boldsymbol{p} \leqslant \boldsymbol{l}$, то, согласно определению оптимального пути, $\boldsymbol{p} = \boldsymbol{l}$. То есть алгоритм находит все оптимальные пути.

2.7. Хранение данных

Для эффективной работы алгоритма необходимо правильно организовать работу с памятью. В нашем случае есть целый ряд множеств, которые должны представлять определённые структуры данных.

Под структурой данных мы понимаем множество операций над этими данными и способ их размещения в памяти. Исследуем примитивные операции, которые мы совершаем над множествами и выберем структуру данных так, чтобы перечисленные операции были бы легко осуществимы, а время выполнения не превосходило бы логарифма от количества элементов во множестве.

Начнём изучение с множества S. Для работы алгоритма необходимо научиться выполнять такие примитивные операции:

1) удалить вершину v: $S^{\text{нов}} = S \setminus \{v\}$;
2) добавить вершину v: $S^{\text{нов}} = S \cup \{v\}$;
3) найти вершину $v \in S$, такую что $\boldsymbol{w}(v) \preceq \boldsymbol{w}(w)$ для всех $w \in S$.

В качестве структуры данных, поддерживающих перечисленные операции, можно взять, например, «очередь с приоритетами», реализованную в виде бинарного дерева. В этом случае время выполнения каждой примитивной операции равно $O(\ln |S|)$, где $|S|$ — размер очереди. Более подробную информацию можно найти в книге [1].

При работе с множествами $\mathbf{P}_v$ необходимо реализовать операцию добавления характеристики $\boldsymbol{p}$: $\mathbf{P}_v^{\text{нов}} = \mathbf{P}_v \cup \{\boldsymbol{p}\}$. Для того чтобы в процессе работы функции $\text{INSERT}(v, \boldsymbol{p})$ можно было быстро определять, существует ли характеристика $\hat{\boldsymbol{p}} \in \mathbf{P}_v$, такая что $\hat{\boldsymbol{p}} \leqslant \boldsymbol{p}$, достаточно проверить выполнение неравенства только в случае последней добавленной характеристики $\hat{\boldsymbol{p}}$ во множество $\mathbf{P}_v$ (см. леммы 2.2, 2.3). Значит, такие множества можно хранить в виде списков и операции, перечисленные выше, будут выполняться за время $O(1)$.

Наиболее сложной представляется работа с множествами $\mathbf{Q}_v$. Перечислим необходимые операции:

1) удалить характеристику $\boldsymbol{q}$: $\mathbf{Q}^{\text{нов}} = \mathbf{Q} \setminus \{\boldsymbol{q}\}$;
2) добавить характеристику $\boldsymbol{q}$: $\mathbf{Q}^{\text{нов}} = \mathbf{Q} \cup \{\boldsymbol{q}\}$;
3) осуществить поиск характеристики $\boldsymbol{q} \in \mathbf{Q}_v$, такой что $\boldsymbol{q} \leqslant \hat{\boldsymbol{q}}$;
4) найти характеристику $\boldsymbol{q} \in \mathbf{Q}_v$, такую что $\boldsymbol{q} \preceq \hat{\boldsymbol{q}}$ для всех $\hat{\boldsymbol{q}} \in \mathbf{Q}_v$.

Множества $\mathbf{Q}_v$ будем хранить в виде сбалансированных деревьев (см. [1]). Тогда все вышеперечисленные операции будут выполняться за время $O(\ln |\mathbf{Q}_v|)$.

2.8. Оценка времени работы и затрат памяти

Для того чтобы оценить трудоёмкость алгоритма, необходимо оценить трудоёмкость работы его составных частей — функций $\mathrm{INSERT}(v, \boldsymbol{p})$ и $\mathrm{EXTRACT}(v, \boldsymbol{p})$. Каждая из этих функций последовательно выполняет примитивные операции, рассмотренные в предыдущем разделе.

Функция INSERT совершает над множеством $\mathbf{Q}_v$ одну операцию добавления и несколько операций поиска с удалением. Количество операций удаления на 1 меньше, чем количество операций поиска, и равно $\Delta\mathbf{Q}_v + 1$, где $\Delta\mathbf{Q}_v = \mathbf{Q}_v^{\text{стар}} - \mathbf{Q}_v^{\text{нов}}$. Поэтому общее количество операций не превосходит $2\Delta\mathbf{Q}_v + 4$. Кроме того, в множество S добавляется вершина v и проверяется принадлежность характеристики $\boldsymbol{p}$ множеству $\mathbf{P}_v$. Соответственно, время работы функции $\mathrm{INSERT}(v, \boldsymbol{p})$ будет

$$O(\ln|S| + (2\Delta\mathbf{Q}_v + 4)\ln|\mathbf{Q}_v|).$$

Функция $\mathrm{EXTRACT}(v, \boldsymbol{p})$ совершает фиксированное количество примитивных операций над каждым из множеств S, $\mathbf{Q}_v$, $\mathbf{P}_v$, поэтому время её работы $O(\ln|S| + \ln|\mathbf{Q}_v|)$.

Обозначим N_v количество оптимальных путей до вершины v. Тогда общее количество оптимальных путей будет равно количеству итераций цикла 4, то есть $N = \sum\limits_{v \in V} N_v$. Количество вызовов функции INSERT для заданной вершины w не превосходит $A_w = \sum\limits_{v \in V} N_v |E_{vw}|$, поэтому $|\mathbf{Q}_w^k| \leqslant A_w$ для всех $k = \overline{1, N}$. Значит, общее время работы функции INSERT для заданной вершины w будет

$$O\biggl(A_w \ln|V| + \sum_i (2\Delta\mathbf{Q}_w^{k_i} + 4)\ln A_w)\biggr),$$

где k_i — номера шагов алгоритма, на которых происходит вызов функции INSERT для заданной вершины w. Так как в начале и в конце работы алгоритма множества $\mathbf{Q}_w$ пусты, получаем, что $\sum\limits_i \Delta\mathbf{Q}_w^{k_i} = 0$. Следовательно, суммарное время работы, затрачиваемое вызовами функции INSERT, будет

$$O\biggl(\sum_{w \in V} (A_w \ln|V| + A_w \ln A_w)\biggr) \qquad (*)$$

Количество вызовов и время работы функции EXTRACT не превосходят соответственно количества вызовов и времени работы функции INSERT, поэтому формула $(*)$ даёт оценку общего времени работы алгоритма. Однако эту формулу не всегда удобно использовать с практической точки зрения.

Обозначим ρ максимальную степень вершины, тогда

$$\ln A_w = \ln\Bigg(\sum_{v\in V} N_v|E_{vw}|\Bigg) \leqslant \ln\Bigg(\rho\sum_{v\in V} N_v\Bigg) = \ln \rho N,$$
$$\sum_{w\in V} A_w = \sum_{w\in V}\sum_{v\in V} N_v|E_{vw}| = \sum_{v\in V} N_v \sum_{w\in V} |E_{vw}| \leqslant \rho N,$$
$$\sum_{w\in V} (A_w \ln|V| + A_w \ln A_w) \leqslant \rho N \ln|V| + \rho N \ln \rho N.$$

Учитывая, что в случае связного графа $|V| \leqslant N$, получаем новую оценку времени работы алгоритма:

$$O(\rho N \ln \rho N) \qquad (**)$$

Рассмотрим несколько полезных примеров использования полученных результатов.

1. Частным случаем данной задачи является задача о нахождении кратчайших путей в графе. Для её решения достаточно положить $\boldsymbol{e}^t = 0$. Тогда $|E_{vw}| \leqslant 1$, $N_v = 1$. Следовательно, $A_w \leqslant |V|$, $\sum\limits_{w\in V} A_w = \sum\limits_{w\in V}\sum\limits_{v\in V} N_v|E_{vw}| = |E|$. Применяя формулу $(*)$, получаем известную оценку $O(|E|\ln|V|)$, рассмотренную в [3].

2. В случае работы с графами дорог степень вершин как правило невелика и чаще всего равна 4. Поэтому формула $(**)$ в этом случае даёт оценку $O(N \ln N)$ времени работы алгоритма.

Объём памяти, необходимый для работы алгоритма, линейно зависит от количества элементов множеств S, $\mathbf{Q}_v$ и $\mathbf{P}_v$.

Недостатки приведённых выше оценок заключаются в том, что заранее невозможно оценить значения параметров N_v.

2.9. Эвристические методы решения

Количество оптимальных путей в графе может быть порядка $2^{|E|}$. Однако с практической точки зрения интерес представляют не все пути, а только те, у которых параметры движения достаточно сильно отличаются, например самый быстрый и самый короткий. В связи с этим возникает необходимость построения алгоритма, который находит «фундаментальное» множество путей малой мощности, близких к оптимальным с различными свойствами.

Исследуем работу функции $\mathrm{INSERT}(v, \boldsymbol{p})$. Характеристику $\boldsymbol{p}$ мы добавляем во множество $\mathbf{Q}_v$ только в том случае, если не существует характеристики $\hat{\boldsymbol{p}} \in \mathbf{P}_v$, такой что $\hat{\boldsymbol{p}} \leqslant \boldsymbol{p}$. Это неравенство означает, что мы уже нашли более короткий и быстрый путь $\hat{p}$, нежели путь p с характеристикой $\boldsymbol{p}$. Однако если $\hat{\boldsymbol{p}} \approx \boldsymbol{p}$, то, по всей видимости, всё равно нет смысла обрабатывать характеристику $\boldsymbol{p}$.

Введём решающее правило $f(v, \boldsymbol{p})$, которое по заданным значениям аргументов и текущим значениям внутренних переменных определяет, стоит ли

рассматривать характеристику $\boldsymbol{p}$. Функцию $\mathrm{INSERT}(v, \boldsymbol{p})$ изменим таким образом, чтобы характеристика $\boldsymbol{p}$ добавлялась во множество $\mathbf{Q}_v$ только в том случае, если $f(v, \boldsymbol{p})$ принимает значение true.

Итак, мы получили модифицированный алгоритм, который выбирает пути в соответствии с критерием f. При удачном выборе критерия можно добиться неплохих результатов и получить общее количество путей порядка $|V|$, а время выполнения алгоритма $O(|E| \ln |V|)$.

В заключение приведём пример функции f:

$$f(v, \boldsymbol{p}) = (\nexists \hat{\boldsymbol{p}} \in \mathbf{P}_v \colon \alpha \hat{\boldsymbol{p}}^d \leqslant \boldsymbol{p}^d),$$

где $\alpha \in [0, 1]$ — параметр критерия.

3. Метод решения задачи нахождения оптимального расписания работы агентов

3.1. Предварительные замечания

При решении оптимизационных задач используются градиентные, генетические и эволюционные методы, эффективность которых во многом зависит от их реализации. Огромное количество публикаций посвящено описанию отдельно взятой реализации какого-либо алгоритма. Наша же основная цель — выделение основных подзадач и описание возможных методов их решения.

Построение расписания можно разбить на три этапа:

1) распределение клиентов между агентами (разбиение множества C на m непересекающихся подмножеств);
2) выбор порядка обслуживания клиентов для каждого агента;
3) оптимизация маршрута каждого агента при фиксированном порядке обслуживания клиентов.

Таким образом, суммарные ограничения по характеристикам для каждого агента проверяются на первом этапе, ограничения на вершины — на первых двух, а на третьем этапе проверяются временные ограничения.

В следующем разделе будут описаны методы построения алгоритмов поиска приближённых решений для первых двух этапов, а в последнем разделе — полиномиальный алгоритм построения оптимального маршрута с любой наперёд заданной точностью.

3.2. Распределение клиентов между агентами и выбор порядка обслуживания

При распределении клиентов между агентами и выборе их порядка обслуживания используются итерационные методы. В данной работе за основу взяты

эволюционные методы. Вначале мы строим колонию (множество приближённых решений), а далее эта колония начинает мутировать, т. е. изменяться по определённым законам. При этом некоторые особи колонии (приближённые решения) исчезают, некоторые появляются, часть особей изменяется. В нашем случае особями колонии являются расписания. В качестве мутаций рассматриваются перестановки частей маршрутов, входящих в расписание, между собой (несколько последовательно обслуживаемых клиентов одного агента и несколько последовательно обслуживаемых клиентов другого агента меняются местами с сохранением порядка обслуживания). Применение таких мутаций позволяет производить быстрый перерасчёт ограничений и значения целевой функции (в случае её аддитивности). Для быстрого определения временны́х ограничений необходимо хранить данные выбора оптимальных рёбер, полученных на третьем этапе решения задачи.

Применяется два вида мутаций: направленные и случайные. В результате применения направленных мутаций значение целевой функции уменьшается. Это позволяет достаточно быстро находить локальный минимум функции. Применение случайных мутаций позволяет выйти из окрестности локального минимума и даёт дополнительные возможности для нахождения глобального минимума целевой функции. Для повышения эффективности алгоритма необходимо согласовывать количество применений направленных и случайных мутаций.

При решении задачи можно использовать две различные стратегии:

1) рассматриваются только корректные расписания;

2) рассматриваются всевозможные расписания.

Первая стратегия предполагает, что на каждом шаге решения задачи рассматриваются только корректные расписания и маршруты. Такое ограничение затрудняет поиск применимых мутаций, но гарантирует построение корректного расписания. В случае, когда пространство корректных расписаний мало, алгоритм может давать достаточно плохие результаты.

Вторая стратегия заключается в том, что мы рассматриваем всевозможные маршруты и изменяем целевую функцию таким образом, что она начинает учитывать степень «некорректности» расписания. Преимущества этого подхода заключаются в том, что мы рассматриваем более широкий класс расписаний и проблем с нахождением применимых мутаций не возникает, однако не гарантируется построения корректного расписания. Ввиду того, что для реальной практической задачи трудно придумать адекватную математическую постановку, довольно часто даже полученные некорректные результаты оказываются удовлетворительными и могут использоваться на практике.

Для начального построения колонии используются различные эвристические методы. Неплохие начальные приближение дают методы имитации течения времени, когда не занятым в данный момент времени агентам назначаются ещё не обслуженные клиенты. Например, всем свободным агентам назначаются клиенты так, чтобы суммарные характеристики маршрутов были бы минимальными.

Такое построения сводится к нахождению паросочетания минимального веса и осуществляется с помощью венгерского алгоритма. В общем случае варьирование стратегий назначения клиентов позволяет получать достаточно большое количество принципиально различных вариантов составления расписания.

После построения начальной колонии производятся следующие действия: применяется серия случайных мутаций, затем серия направленных мутаций, опять серия случайных мутаций и т. д. Общее количество итераций обычно определяется допустимым временем работы программы, так как чаще всего теоретические оценки неприменимы.

3.3. Оптимизация маршрута отдельного агента

Преобразованный граф G' имеет достаточно большое количество кратных рёбер за счёт того, что существуют несоизмеримые пути в графе G. Соответственно, правильный выбор рёбер отдельного маршрута даёт возможность уменьшить значение целевой функции. Ниже приводится алгоритм построения достаточно хорошего приближения.

Формализуем задачу. Задан порядок обслуживания клиентов, т. е. последовательность вершин $v_0 \dots v_l \in V'$, входящих в маршрут агента. Обозначим $e_i^1 \dots e_i^{k_i} \in E'$ всевозможные рёбра, соединяющие вершину v_{i-1} и вершину v_i. На посещение каждой из вершин заданы временны́е ограничения, которые учитывают возможность добраться до заданной вершины и обслужить клиента. Скорректируем временны́е характеристики рёбер так, чтобы автоматически учитывалось время на обслуживание клиентов. Тогда временны́е ограничения примут вид $t_i \in (\mathbf{c}^\tau)_i$ и $(e^t)_i \leqslant t_i - t_{i-1}$, где $e_i \in \{e_i^1 \dots e_i^{k_i}\}$ — i-е ребро маршрута. Будем считать, что длина рёбер означает вклад в общее значение целевой функции для данного маршрута (остальные маршруты расписания фиксированы). Тогда нам необходимо минимизировать значение суммы $\sum\limits_{i=1}^{n} (e^d)_i$.

Рассмотрим дискретизацию времени с некоторым фиксированным шагом Δt. Для каждой вершины v_i, $i = \overline{1,l}$, и всех моментов времени $n\Delta t$ найдём оптимальный маршрут из начальной вершины v_0, удовлетворяющий временны́м ограничениям. Обозначим длину этого маршрута $f(i,n)$. Если не существует маршрута, удовлетворяющего временны́м ограничениям, будем считать, что функция не определена. Значение функции $f(i,n)$ динамически пересчитывается по значениям $f(i-1,k)$ (перебираем поочерёдно рёбра $e_i^1 \dots e_i^{l_i}$). Поскольку функция невозрастающая по второму аргументу, для пересчёта значений функции достаточно каждое ребро просмотреть не более одного раза. Если функция $f(i,n)$ определена, то n удовлетворяет условию $n\Delta t \in [t_{i-1}, t_i]$. Поэтому общее время работы этого алгоритма $O(N_v N_t + N_e)$, где $N_v = l$ — количество вершин в маршруте, N_t — количество интервалов времени работы агента, $N_e = k_1 + \ldots + k_l$ — общее количество рассматриваемых рёбер.

Уменьшение шага дискретизации Δt позволяет находить решение задачи с любой наперёд заданной точностью.

Заключение

В работе описано несколько различных подходов для построения алгоритмов поиска приближённых решений оптимизационных задач на графах больших размерностей. Для ряда подзадач приведены алгоритмы нахождения точных решений. Описаны возможные эвристические подходы для повышения общей скорости работы.

Алгоритмы, приведённые в данной работе, применялись на практике и дали достаточно неплохие результаты. Тестирование проводилось на множестве практических задач, в том числе и на задачах, описанных в начале работы. В качестве сеток дорог рассматривались реальные карты — улицы города Москвы и дороги Бельгии. Самый большой граф, на котором проводилось тестирование, содержал 3036148 вершин, 4149025 рёбер и 3720 точек обслуживания. С результатами этих экспериментов можно познакомится в работе [5].

Алгоритмы, приведённые в данной работе, достаточно хорошо распараллеливаются и дают высокий коэффициент утилизации при увеличении числа процессоров. Результаты тестирования можно посмотреть в работе [4].

Недостатком вышеизложенных методов является тот факт, что заранее невозможно предсказать результаты работы программы. К сожалению, бывают случаи, когда результаты неудовлетворительны. Наличие таких результатов связано с тем фактом, что задачи имеют большую вычислительную сложность. Построение универсального алгоритма невозможно в предположении выполнения гипотезы о том, что класс $\mathbf{P}$ полиномиально-вычислимых задач не совпадает с классом $\mathbf{NP}$-полных задач, т. е. $\mathbf{P} \neq \mathbf{NP}$.

Литература

[1] Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. — М.: МЦНМО, 1999.

[2] Thorup M. Undirected single-source shortest paths with positive integer weights in linear time // Journal of the ACM. — 1999. — Vol. 46, no. 3. — P. 362—394.

[3] Williams J. W. J. Heapsort // Commun. ACM. — 1964. — Vol. 7, no. 6 (June). — P. 347—348.

[4] Инюхин А. В., Панкратьев Е. В., Чеповский А. М., Чернышев С. В. Использование Т-системы для преобразования графа дорог в задаче оптимизации маршрутов движения // Высокопроизводительные вычисления и их приложения: Труды Всероссийской научной конференции (30 октября – 2 ноября 2000 г., г. Черноголовка). — М.: Изд-во Моск. ун-та, 2000. — С. 220—223.

[5] Панкратьев Е. В., Чеповский А. М., Черепанов Е. А., Чернышев С. В. Нахождение наборов оптимальных маршрутов на больших сетках дорог геоинформационных систем // Проблемы передачи и обработки информации в сетях и системах телекоммуникаций: Материалы 10-й Международной науч.-техн. конф. — Рязань: Рязанская государственная радиотехническая академия, 2001. — С. 240—241.