

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра дискретной математики и алгоритмики

Касияник Алексей Леонидович

**МНОГОКРИТЕРИАЛЬНАЯ МНОГОМОДАЛЬНАЯ ЗАДАЧА
ПЛАНИРОВАНИЯ ТУРИСТИЧЕСКОГО МАРШРУТА**

Отчет по практике

Специальность 1-31 81 09 «Алгоритмы и системы обработки больших объемов
информации»

Руководитель практики от кафедры
Соболевская Елена Павловна
доцент, кандидат физико-математических
наук

Руководитель практики от организации
Михальчук Дмитрий Викторович
руководитель проекта

Минск, 2017

Реферат

Дипломная работа, 46 с., 29 рис., 8 источников, 1 приложение.

МНОГОКРИТЕРИАЛЬНАЯ ЗАДАЧА ОПТИМИЗАЦИИ,
МНОГОМОДАЛЬНАЯ ЗАДАЧА МАРШРУТИЗАЦИИ, ПОСТРОЕНИЕ
ОПТИМАЛЬНОГО МАРШРУТА, JAVA, CPLEX.

Объектом исследования являются многокритериальные маршруты для городского туризма.

Целью работы является разработка приложения, позволяющего строить многокритериальные многомодальные маршруты.

В результате работы было разработано приложение, позволяющее строить многокритериальные многомодальные маршруты по городу Минск.

Методы исследования: изучение литературы, методы оптимизации, теория алгоритмов.

Область применения: построение маршрутов для городского туризма.

Рэферат

Дыпломная праца, 46 с., 1 мал., 8 крыніц, 1 дадатак.

ШМАТКРЫТЭРЫАЛЬНАЯ ЗАДАЧА АПТЫМІЗАЦЫ,
ШМАТМАДАЛЬНАЯ ЗАДАЧА МАРШРУТЫЗАЦЫ, ПАБУДОВА
АПТЫМАЛЬНАГА МАРШРУТА, JAVA, CPLEX.

Аб'ектам даследавання з'яўляюцца шматкрытэрыальныя маршруты для гарадскога турызму.

Мэтай працы з'яўляецца распрацоўка прыкладання, якое дазваляе будаваць шматкрытэрыальныя шматмадальныя маршруты.

У выніку працы было распрацава прыкладанне, якое дазваляе будаваць шматкрытэрыальныя шматмадальныя маршруты па горадзе Мінску.

Метады даследавання: вывучэнне літаратуры, метады аптымізацыі, тэорыя алгарытмаў.

Вобласць выкарыстання: пабудова маршрутаў для гарадскога турызма.

Abstract

Graduation work, 46 pp., 1 pictures, 8 sources, 1 appendix.

MULTI-OBJECTIVE PATH PROBLEM, MULTIMODAL PATH PROBLEM,
OPTIMAL ROUTE, JAVA, CPLEX.

Object of research: multi-objective routes for city tourism.

Goal of the research: development of the application for multi-objective routes construction.

The result of current research is the application for multi-objective routes construction within Minsk city.

Research methods: analysis of topic relevant literature, optimization theory, algorithms theory.

Applications: routes construction for city tourism.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	6
ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ	8
ГЛАВА 2 МНОГОКРИТЕРИАЛЬНАЯ ЗАДАЧА ОПТИМИЗАЦИИ	10
2.1 Постановка задачи	10
2.2 Методы решения	11
2.2.1 Лексикографическое упорядочение критериев.....	11
2.2.2 Метод последовательных уступок по значению ведущего критерия.....	12
2.2.3 Метод главного критерия.....	13
2.2.4 Свертка критериев	13
ГЛАВА 3 ПОСТРОЕНИЕ МОДЕЛИ И АЛГОРИТМА РЕШЕНИЯ	16
3.1 Общее описание подхода	16
3.2 Моделирование сети	17
3.2.1 «Изменяющийся во времени» граф.....	17
3.2.2 Транспортные модули	18
3.2.2 Модуль пересадки.....	19
3.2.3 Модуль посещения	19
3.2.4 Веса дуг	20
3.3 Метод решения.....	21
3.4 Формулирование задач булевского программирования	23
3.4.1 Максимизация количества посещенных точек интереса	23
3.4.2 Минимизация времени прибытия в конечную точку маршрута	25
3.4.3 Минимизация стоимости путешествия.....	26
3.4.4 Минимизация количества пересадок	26
3.4.5 Минимизация общего времени пересадок и ожидания	27
3.4.6 Минимизация общего времени в пешем модуле	27
3.4.7 Минимизация выброса CO ₂	27
ГЛАВА 4 РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТРОЕНИЯ МАРШРУТОВ	29
4.1 Входные параметры	29
4.2 Архитектура приложения.....	30
4.3 Средства разработки	31
4.3.1 Java	31
4.3.2 CPLEX	31
Список использованных источников.....	32
ПРИЛОЖЕНИЕ А	33

ВВЕДЕНИЕ

Автоматическое создание городских туристических маршрутов является интересной и востребованной задачей, которая приобретает особенную важность в контексте современных требований и вызовов. Современный пользователь имеет широкий выбор решений, предлагающий сервис планирования путешествий и маршрутов, однако большинство таких приложений имеют весьма ограниченный функционал: например, учитывают (или не учитывают вовсе) только какой-то конкретный вид транспорта. Но особенно важно то, что существующие решения практически полностью игнорируют ключевой функционал – многокритериальную оптимизацию.

Задача многокритериального многомодального построения туристических маршрутов особенно интересна в контексте стремительно растущей проблемы трафика на дорогах, имеющей значительное влияние на экономику и окружающую среду. Операторы трафика рассматривают использование многомодального транспорта (общественный транспорт, велосипед, park'n'ride и т.д.) в качестве хорошей стратегии решения проблем перегруженности и пагубного воздействие на окружающую среду.

Многомодальный многокритериальный планировщик путешествий предоставляет пользователю различные варианты маршрутов, оптимизированных по критериям, которые предпочитает пользователь (общая стоимость путешествия, скорость и время, количество мест для посещения, экологичность).

Построение многомодальных и многокритериальных маршрутов само по себе является сложной математической задачей, которая существенно усложняется необходимостью обработки и интеграции больших объемов данных широкого спектра, в том числе информацию о дорожной сети, точках интереса (POI), расписания общественного транспорта, а также необходимостью оптимизации для конкурирующих критериев, где полная оптимизация для одного критерия, например, время в пути, может отрицательно повлиять на другие критерии, например, стоимость и экологичность (выброс CO в атмосферу). Взаимосвязь между такими критериями может быть иметь весьма

субъективный характер, так как их приоритеты будут варьироваться от пользователя к пользователю.

В рамках данной работы ставится задача разработки приложения для создания многомодальных многокритериальных маршрутов для городского туризма.

ГЛАВА 1

ПОСТАНОВКА ЗАДАЧИ

Многомодальный планировщик путешествий – эффективный инструмент при планировании путешествий из одной точки в другую, используя различные виды транспорта. Однако анализ существующих многомодальных планировщиков показывает, что ключевая составляющая – многокритериальная оптимизация – часто отсутствует. Целью данной работы является разработка приложения, которое бы позволяло пользователю находить наилучший маршрут, учитывая не только продолжительность путешествия, но и такие важные для современного человека критерии, как выброс CO_2 , расстояние, пройденное пешком, стоимость передвижения, и т.д.

Будем полагать, что турист планирует свою поездку в каком-то временном окне и в определенной географической области. Пользователь должен иметь возможность отправить свой запрос приложению-планировщику, включая следующие параметры: *регион* путешествия (в данной работе будем рассматривать города), точку отправления i^* , точку прибытия j^* , минимальное время отправления s^* , максимальное время прибытия t^* , список *точек интереса*, которые пользователь хочет посетить, а также удобные для него способы передвижения, или, другими словами, *транспортные модули*. В качестве транспортных модулей предполагается рассмотреть общественный транспорт (автобус, метро, и т.д.), транспорт без фиксированного расписания (такси), велосипед и пешие прогулки. Транспортные модули могут быть заданы в любой комбинации.

Также пользователь должен иметь возможность выбрать маршрут, который соответствовал бы некоторым критериям. В качестве таких критериев рассмотрим следующие:

1. максимизация количества посещенных точек интереса (POI),
2. минимизация времени прибытия в конечную точку маршрута,
3. минимизация стоимости путешествия,
4. минимизация количества изменений транспортных модулей,
5. минимизация общего времени пересадок и ожидания,
6. минимизация общего времени в пешем модуль,

7. минимизация выбросов CO₂.

Предполагается, что критерии будут упорядочены пользователем по степени важности, и менее важный критерий будет оптимизирован на множество близких к оптимальному решению более важного критерия.

Маршрут может проходить через одну и ту же географическую точку несколько раз, а также допускается, что точки i^* и j^* являются одной и той же точкой. Точной отправления и точкой прибытия могут являться точки интереса.

ГЛАВА 2

МНОГОКРИТЕРИАЛЬНАЯ ЗАДАЧА ОПТИМИЗАЦИИ

2.1 Постановка задачи

Многокритериальная задача оптимизации может быть записана в следующем виде:

$$\min_{x \in D} (Q_1(x), Q_2(x), \dots, Q_m(x)) \quad (2.1.1)$$

где $Q_i: R^n \rightarrow R$ – целевые функции, которые могут конфликтовать друг с другом, и $D \in R^n$ – область допустимых решений.

Задачи максимизации можно привести к задачам минимизации следующим образом:

$$\max Q_i \rightarrow \min(-Q_i) \quad (2.1.2)$$

Цель многокритериальной задачи оптимизации – одновременно минимизировать все целевые функции.

Обозначим за $D \subset R^m$ отображение пространства решений в область допустимых целевых значений, т.е. каждому элементу пространства решений сопоставим вектор из значений целевых функций на этом элементе:

$$Z = \{z \in R^m : z = (Q_1(x), Q_2(x), \dots, Q_m(x)), \forall x \in D\} \quad (2.1.3)$$

Так как мы предполагаем, что целевые функции могут конфликтовать друг с другом и не являются независимыми, то может не существовать единственного решения, которое сможет оптимизировать все целевые функции одновременно. Поэтому под оптимальным решением будем понимать решение, в котором невозможно улучшить значение одной целевой функции без ухудшения другой целевой функции.

2.2 Методы решения

2.2.1 Лексикографическое упорядочение критериев

Рассмотрим схему компромисса – лексикографическое упорядочение критериев (ЛУК). Данная схема предусматривает, что последовательность, в которой критерии перечислены, определяет их значимость в следующем смысле: каждый предшествующий критерий несравненно важнее любого из перечисляемых за ним. Синтез решения, оптимального при лексикографическом упорядочении критериев, реализуется следующим образом.

Сначала решаем однокритериальную задачу

$$\min_{x \in D} Q_1(x) \quad (2.2.1)$$

Обозначим D_1 множество ее оптимальных решений. Если D_1 – одноэлементное множество, то единственное оптимальное решение задачи (2.2.1) является одновременно оптимальным по принципу ЛУК исходной многокритериальной задачи (2.1.1). В противном случае далее решаем задачу

$$\min_{x \in D_1} Q_2(x) \quad (2.2.2)$$

Обозначим D_2 множество оптимальных решений задачи (2.2.2). Если D_2 – одноэлементное множество, то единственное оптимальное решение задачи (2.2.2) является одновременно оптимальным по принципу лексикографического упорядочения критериев решением исходной многокритериальной задачи (2.1.1). В противном случае поступаем аналогично предыдущему – решаем задачу минимизации значения критерия $Q_3(x)$ при условии, что $x \in D_2$, и т.д. Максимальное число последовательно решаемых однокритериальных задач равно m , т.е. числу критериев исходной многокритериальной задачи. Если решение задачи (2.1.1) определилось в результате выполнения меньшего числа этапов, мы остановились на k -ой итерации, то оно единственno. В противном случае может оказаться, что оптимальных (при имеющемся лексикографическом упорядочении критериев) решений этой задачи более чем одно; все они эквивалентны.

2.2.2 Метод последовательных уступок по значению ведущего критерия

Суть метода последовательных уступок по значению ведущего критерия поясним сначала на примере бикритериальной задачи максимизации

$$\max_{x \in D_1} (Q_1(x), Q_2(x)) \quad (2.2.3)$$

в которой критерий $Q_1(x)$ считаем ведущим. Реализуя этот метод, сначала находим решение x^* , оптимальное при лексикографическом упорядочении критериев $(Q_1(x), Q_2(x))$. Пусть $(Q_1(x^*), Q_2(x^*)) = (a, b)$. Точка (a, b) – оценка найденного решения. Если данная оценка (первая ее координата a – максимально возможное значение критерия $Q_1(x)$, а вторая координата b – максимально возможное значение критерия $Q_2(x)$) удовлетворяет лицо, принимающее решение, то x^* принимается за искомое оптимально-компромиссное решение; в противном случае лицо, принимающее решение (ЛПР), назначает уступку $\delta_1, \delta_1 > 0$ по допустимому значению первого критерия. Далее решается задача

$$\max_{x \in D_1} Q_2(x) \quad (2.2.4)$$

при дополнительном условии

$$Q_1(x) \geq a - \delta_1 \quad (2.2.5)$$

Пусть оптимальное решение x^{**} задачи в исходной задаче (2.2.3) имеет оценку (a_1, b_1) . Очевидно, $b_1 \geq b$. Если данная оценка удовлетворяет ЛПР, то x^{**} принимается за искомое оптимально-компромиссное решение задачи (2.2.3) в противном случае ЛПР назначает новую уступку δ_2 , где $\delta_2 > \delta_1$, по допустимому значению первого критерия. Далее решается задача

$$\max_{x \in D} Q_2(x) \quad (2.2.6)$$

при дополнительном условии

$$Q_1(x) \geq a - \delta_2 \quad (2.2.7)$$

Пусть оптимальное решение x^{***} задачи (2.2.6) - (1.2.7) в исходной задаче (2.2.3) имеет оценку (a_2, b_2) . Как очевидно, $b_2 \geq b$. Если данная оценка удовлетворяет ЛПР, то x^{***} принимается за искомое оптимально-компромиссное решение задачи (2.2.3); в противном случае ЛПР назначает новую уступку δ_3 , $\delta_3 > \delta_2$ по допустимому значению первого критерия, и т.д.

Описанный процесс продолжается вплоть до отыскания решения с оценкой, устраивающей лицо, принимающее решение, или вплоть до ситуации, когда дальнейшие уступки по допустимому значению первого критерия становятся невозможными.

2.2.3 Метод главного критерия

Метод главного критерия заключается в оптимизации значения наиболее важного критерия при условии, что остальные критерии принимают значения, не больше предписанных пороговых величин. Вводим нумерацию, при которой $Q_1(x)$ - главный критерий. Тогда задача (2.2.1) сводится к однокритериальной задаче

$$\min_{x \in D} Q_1(x) \quad (2.2.8)$$

при дополнительных условиях

$$Q_k \leq h_k, k = 2, 3, \dots, m \quad (2.2.9)$$

где h_k – заданные соответственно для второго, третьего, ..., m -го критериев пороги.

2.2.4 Свертка критериев

Метод линейной свертки позволяет перейти от многокритериальной задачи к однокритериальной задаче оптимизации. Целевая функция в последней является суммой целевых функций Q_i , домноженных на весовые коэффициенты. Эти коэффициенты могут быть нормализованы, чтобы их сумма давала единицу, что не обязательно в общем случае.

Задачу линейной свертки критериев можно записать следующим образом

$$K(x) = \lambda_1 Q_1(x) + \lambda_2 Q_2(x) + \dots + \lambda_m Q_m(x) \rightarrow \min \quad (2.2.10)$$

при условии $x \in D$. При этом коэффициенты имеют следующие ограничения:

$$\begin{aligned} \lambda_i &\in (0, 1), \quad i = \overline{1, m}, \\ \lambda_1 + \lambda_2 + \dots + \lambda_m &= 1 \end{aligned} \quad (2.2.11)$$

Так как целевые функции могут иметь различную степень важности, а также иметь различные промежутки значений, это значит, что нам может понадобиться нормализация целевых функций для получения оптимального по Парето решения. В качестве нормализующей схемы можно использовать нормализацию по разнице значений целевых функций в точках Надир (N) и Утопия (U), которая даст нам длину интервала, где оптимальное значение целевой функции может варьироваться в пределах Парето-фрона.

Можно заметить, что размеры множества Парето-оптимальных альтернатив предоставляют информацию для решения задачи. Компоненты $z_i^* = Q_i(x^i) \in R$ идеального целевого вектора $z^* \in R^k$ определяются минимизацией каждой целевой функции независимо от остальных, т. е.:

$$z_i^* = Q_i(x^i), \quad x^i = \arg \min Q_i(x) \quad (2.2.12)$$

Идеальный вектор $z^U = z^*$ называемый точкой Утопия, не всегда может быть допустимым, так как могут существовать конфликты между целевыми функциями. Данная точка дает нам информацию о нижней границе Парето-фрона.

Верхние границы Парето-фрона определяются компонентами точки Надира:

$$z_i^N = \max_{1 \leq j \leq k} Q_i(x^j), i = \overline{1, k} \quad (2.2.13)$$

Так как мы нормализуем целевые функции по интервалам их изменения в Парето-фронте, то для улучшения результатов нормализации мы будем использовать следующие коэффициенты:

$$\theta_i = \frac{1}{z_i^N - z_i^U} \quad (2.2.14)$$

Можно заметить, что после нормализации целевые функции будут ограничены:

$$0 \leq \frac{Q_i(x) - z_i^U}{z_i^N - z_i^U} \leq 1 \quad (2.2.15)$$

что дает одинаковый интервал для изменения целевых функций.

Для вычисления нормализационного интервала $z_i^N - z_i^U$ необходимо решить к задач оптимизации вида $\min_{x \in D} Q_i(x)$ для получения x^i . Значение этих элементов является ключевым, так как они необходимы для вычисления $z_i^U = Q_i(x^i)$ и $z_i^N = \max_j Q_i(x^j)$.

ГЛАВА 3 ПОСТРОЕНИЕ МОДЕЛИ И АЛГОРИТМА РЕШЕНИЯ

3.1 Общее описание подхода

Представленный ниже подход основывается на приблизительной лексикографической оптимизации. Критерии, по которым строится маршрут, предварительно ранжируются в порядке важности. Далее находится оптимальный маршрут относительно первого наиболее важного критерия. Следующим шагом является поиск наилучшего маршрута относительно второго по значимости критерия таким образом, что отклонение от значения первого критерия находится в заданном промежутке. Далее происходит поиск оптимального маршрута по третьему критерию таким образом, что отклонения от значений первого и второго критериев находятся в заданных допустимых промежутках. Следующие шаги строятся по аналогии.

Данный подход к решению многокритериальной задачи представляется хорошим компромиссом между оригинальным лексикографическим подходом, подходом свертки всех критериев в один и подходом с независимыми критериями.

3.2 Моделирование сети

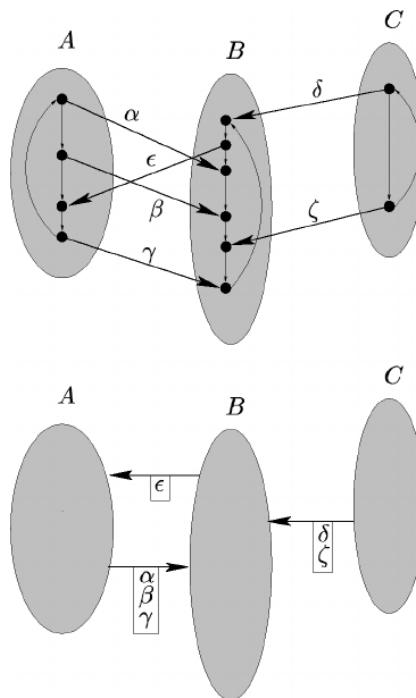
3.2.1 «Изменяющийся во времени» граф

Существует два подхода к моделированию пространственных данных с учетом зависимости от времени:

- построение «развернутого во времени» графа (time-expanded approach)
- построение «изменяющегося во времени» графа (time-dependent approach).

При построении «развернутого во времени» графа каждая вершина рассматривается как пара (l, t) , где l – географическая локация, t – время события. Событиями являются все отправления и прибытия транспорта в данную локацию. Например, в данной модели все отправления поездов с вокзала будут рассматриваться как самостоятельные вершины в графе.

При построении «изменяющегося во времени» графе вершины не ассоциируются с временем и будут представлять только локацию.



1. Рис. 1 “Развернутый во времени” граф (сверху) и “изменяющийся во времени” граф

2.

Построим модель взвешенной сети используя «изменяющийся во времени» граф

$$G = (N, A),$$

где

N – множество вершин,

A – множество ребер.

Вершины представлены в виде пар (l, t) , где $l \in L$ – географическая локация, t – время события.

Обозначим через L множество географических точек (локаций):

- точки интереса,
- вокзалы,
- остановки общественного транспорта с фиксированным расписанием,
- места, в которых турист может воспользоваться транспортом без фиксированного расписания (например, парковки такси).

Две вершины (i, a) и (j, b) соединены ребром (m, i, a, j, b) тогда и только тогда, если турист может отправиться из точки i в момент времени a и прибыть в точку j в момент времени b , используя транспортный модуль (вид транспорта) m . Таким образом, если несколько видов транспорта позволяют выехать из точки i в момент a и прибыть в j в момент b , то вершины будут соединены несколькими параллельными дугами, отличающимися в m .

3.2.2 Транспортные модули

Дуга (m, i, a, j, b) для транспортного модуля с фиксированным расписанием будет существовать в графе тогда и только тогда, если в соответствии с расписание транспорта m пассажир может отправиться из точки i в момент времени a и прибыть в точку j в момент времени b .

Для транспорта без фиксированного расписания будем полагать, что такая дуга будет существовать тогда и только тогда, если $b = a + \left\lceil \frac{l_{ij}}{v_{ijm}} \right\rceil$ и перемещение из точки i в точку j во временном интервале $[a, b]$ возможно в соответствии с правилами дорожного движения. Для таких типов транспорта маршрут передвижения между точками должен быть предварительно задан. Подразумевается, что данный путь является кратчайшим путем и может быть найден с помощью одного из известных алгоритмов поиска кратчайших путей, например, алгоритмом Флойда-Уоршелла или алгоритмом Джонсона.

3.2.2 Модуль пересадки

Разрабатываемая модель должна предусматривать ситуацию, когда турист должен задержаться в каком-то месте на определенное время. Например, если локация является музеем, то возможно туристу придется подождать открытия. В случае, если точка является вокзалом, то модель должна описывать время ожидания пересадки. Для этого введем дополнительный модуль *пересадки* t . В таком случае будем полагать, что вершины (i, a) и (i, b) соединены дугой *пересадки* $(t, i, a, i, b), a < b$.

3.2.3 Модуль посещения

Точки интереса (англ. point of interest, POI) – это особые точки на карте, которые могут быть кому-то полезны или интересны. В нашем случае, под точками интереса будем подразумевать различные объекты на местности, которые могут быть интересны с точки зрения туризма. Это могут быть музеи, галереи, театры, памятники архитектуры и прочие достопримечательности.

Пусть P – множество точек интереса и $P \subset L$. Каждой точке $i \in P$ присвоим временной интервал $[O_i, C_i]$, который определяет время, когда турист может посетить точку i . Например, интервал $[O_i, C_i]$ может соответствовать времени работы музея. *Продолжительность посещения* d_i точки интереса i будет задаваться пользователем.

Чтобы смоделировать возможность посещения туристом POI, введем дополнительный модуль – модуль *посещения* v и положим, что вершины (i, a) и (i, b) соединены дугой посещения (v, i, a, j, b) тогда и только тогда, когда $a < b$, $i \in P$, $[a, b] \subseteq [O_i, C_i]$ и $a - b = d_i$.

3.2.4 Веса дуг

Каждая дуга (m, i, a, j, b) будет иметь следующее множество весов:

- l_{ij} – расстояние между вершинами, которое будет пройдено используя режим m ;
- v_{ijm} – средняя скорость перемещения для режима m на данном участке. Для транспорта с фиксированным расписанием v_{ijm} будем полагать равной рекомендованной скорости. Для пешего и велосипедного режима $v_{ijm} \in \{v_m^{(s)}, v_m^{(h)}\}$, где $v_m^{(s)}$ – стандартная средняя скорость, $v_m^{(h)}$ – высокая средняя скорость;
- $p_{m,i,a,j,b}$ – стоимость перемещения при использовании модуля m на данном участке ($p_{m,i,a,j,b} = 0$ – для пешего и велосипедного режима и режима ожидания). Стоимость перемещения может включать стоимость проезда общественным транспортом, стоимость затраченного топлива в случае перемещения с помощью личного автомобиля и т.д.;
- $c_{m,i,a,j,b}$ – количество выбрасываемого CO₂ ($c_{m,i,a,j,b} = 0$ – для пешего, велосипедного модуля и модулей пересадки и посещения).

3.3 Метод решения

Введем обозначения для параметров, которые будут использоваться при моделировании критериев маршрута, в терминах смоделированной в предыдущей главе сети:

1. $i^* \in L$ – точка отправления,
2. $j^* \in L$ – точка прибытия,
3. s^* – самое раннее допустимое время отправления,
4. t^* – самое позднее допустимое время прибытия,
5. $\{Q_k \mid k = 0, \dots, n\}$ – множество критериев, упорядоченных по убыванию степени важности,
6. $\{\varepsilon_k \mid k = 0, \dots, n\}$ – упорядоченное множество отклонений для значений критериев,
7. P – множество точек интереса, которые предполагается посетить,
8. d_i – длительность посещения точек P
9. M – множество допустимых транспортных модулей,
10. v_w – средняя скорость перемещения пешком,
11. v_b – средняя скорость перемещения на велосипеде.

Метод решения основывается на методе лексикографического упорядочения критериев. Данный подход расширен с помощью введения относительного отклонения ε_k от значения F_k решения задачи булевского линейного программирования (БЛП) для критерия Q_k . Моделирование задач БЛП приведено в следующем разделе.

Пусть $\sigma_k = 1 + \varepsilon_k$, если Q_k – критерий минимизации, и $\sigma_k = 1 - \varepsilon_k$, если Q_k – критерий максимизации. Решением задачи будет являться множество решений $x^{(1)}, \dots, x^{(k)}$, полученных следующим образом:

1. $x^{(1)}$ – оптимальное решение задачи БЛП для первого критерия Q_1 на множестве всех допустимых маршрутов со значением $F_1 = Q_1(x^{(1)})$;

2. $x^{(2)}$ – оптимальное решение задачи БЛП для второго критерия Q_2 со значением $F_2 = Q_2(x^{(2)})$, при этом $Q_1(x^{(2)}) \leq \sigma_1 F_1$;
3. $x^{(3)}$ – оптимальное решение задачи БЛП для третьего критерия критерия Q_3 со значением $F_3 = Q_3(x^{(3)})$, при этом $Q_1(x^{(3)}) \leq \sigma_1 F_1$ и $Q_2(x^{(3)}) \leq \sigma_2 F_2$;

и т.д.

Следует заметить, что только решение $x^{(1)}$ является оптимальным относительно первого критерия. Другие решения не обязательно являются оптимальными относительно соответствующих критериев, поскольку соответствующая область поиска не включает все возможные маршруты. Однако решения $x^{(k)}, k > 1$ могут рассматриваться как близкие к лексикографическому оптимуму для упорядоченных критериев Q_k .

3.4 Формулирование задач булевского программирования

В данной главе опишем формулирование задач булевского линейного программирования для модели сети, описанной в предыдущей главе. Предположим, что пользователь выбрал все возможные критерии и упорядочил их следующим образом:

1. максимизация количества посещенных точек интереса (POI),
2. минимизация времени прибытия в конечную точку маршрута,
3. минимизация стоимости путешествия
4. минимизация количества изменений транспортных модулей,
5. минимизация общего времени пересадок и ожидания,
6. минимизация общего времени в пешем модуль,
7. минимизация выброса CO_2 .

Составление задач для иного порядка критериев происходит по аналогичной схеме.

3.4.1 Максимизация количества посещенных точек интереса

Задачу булевского программирования для максимизации количества посещенных точек сформулируем следующим образом:

$$\sum_{i \in I} \sum_{(v,i,a,i,b) \in V} x_{v,i,a,i,b} \rightarrow \max, \quad (3.4.1)$$

где

I – множество точек POI, выбранных пользователем для посещения,

V – множество дуг посещения.

Сформулируем условия для данной задачи.

Маршрут должен содержать в точности одну дугу, выходящую из начальной вершины (i^*, s^*) :

$$\sum_{(m,i^*,s^*,j,b) \in A} x_{m,i^*,s^*,j,b} = 1 \quad (3.4.2)$$

Маршрут должен содержать в точности одну дугу, входящую в конечную вершину (j^*, t^*) :

$$\sum_{(m,i,a,j^*,t^*) \in A} x_{m,i,a,j^*,t^*} = 1 \quad (3.4.3)$$

Нужно гарантировать, что все промежуточные вершины пути посещаются единожды. Для каждой точки маршрута не может быть больше одной входящей и выходящей дуги:

$$\sum_{(m,i,a,j,b) \in A} x_{m,i,a,j,b} \leq 1, (j, b) \in N \quad (3.4.4)$$

$$\sum_{(m,i,a,j,b) \in A} x_{m,i,a,j,b} \leq 1, (i, a) \in N \quad (3.4.5)$$

Количество входящих и выходящих дуг из промежуточных вершин маршрута должно быть равно:

$$\begin{aligned} \sum_{(m,i,a,h,c) \in A} x_{m,i,a,h,c} &= \sum_{(m,h,c,j,b) \in A} x_{m,h,c,j,b}, \\ (h, c) \in N, \end{aligned} \quad (3.4.6)$$

$$c = s^* + 1, \dots, t^* - 1$$

Не более одной дуги пересадки для каждой вершины пути:

$$\sum_{(t,i,a,i,b) \in T} x_{t,i,a,i,b} + \sum_{(t,i,b,i,c) \in T} x_{t,i,b,i,c} \leq 1, (i, b) \in N \quad (3.4.7)$$

Каждая точка интереса посещается не более одного раза:

$$\sum_{(\nu,i,a,i,b) \in V} x_{\nu,i,a,i,b} \leq 1, i \in I \quad (3.4.8)$$

В начальный момент времени s^* дуги выходят только из стартовой точки i^* :

$$\sum_{\substack{(m,i,s^*,j,b) \in A, \\ i \neq i^*}} x_{m,i,s^*,j,b} = 0 \quad (3.4.9)$$

Переход между модулями происходит только посредством трансферной дуги:

$$\begin{aligned} \sum_{(m,i,a,h,c) \in A \setminus \{T \cup V\}} x_{m,i,a,h,c} + \sum_{(m',h,c,j,b) \in A \setminus \{T \cup V\}} x_{m',h,c,j,b} &\leq 1 \\ (h, c) \in N, \\ m, m' \in M, m \neq m' \end{aligned} \quad (3.4.10)$$

$$x_{m,i,a,j,b} \in \{0,1\}, (m, i, a, j, b) \in A \quad (3.4.11)$$

Пусть P^* – значение оптимального решения задачи максимизации количества посещенных точек. Следующую задачу булевского программирования сконструируем для критерия минимизации времени прибытия.

3.4.2 Минимизация времени прибытия в конечную точку маршрута

Целевая функция для данной задачи будет иметь следующий вид:

$$\sum_{\substack{(m,i,a,j^*,t^*) \in A, \\ m \neq r}} t^* \cdot x_{m,i,a,j^*,t^*} + \sum_{(r,j^*,a,j^*,t^*) \in A} a \cdot x_{r,j^*,a,j^*,t^*} \rightarrow \min \quad (3.4.12)$$

В данной модели r – искусственный модуль, который используется для перемещения туриста в финальной точке из любого момента времени $a, s^* \leq a < t$ в момент времени t^* .

К условиям (3.4.1) – (3.4.11) добавим ограничение на количество точек для посещения так, чтобы пользователь по меньшей мере посетил $\lfloor P^*(1 - \varepsilon_P) \rfloor$ точек, где ε_P – относительное отклонение количества посещенных POI от оптимума:

$$\sum_{i \in I} \sum_{(v,i,a,i,b) \in V} x_{v,i,a,i,b} \geq \lfloor P^*(1 - \varepsilon_P) \rfloor \quad (3.4.13)$$

Значение оптимального решения данной задачи обозначим через D^* .

3.4.3 Минимизация стоимости путешествия

Целевая функция для данной задачи будет иметь следующий вид:

$$\sum_{(m,i,a,j,b) \in A \setminus (T \cup U \cup R)} p_{m,i,a,j,b} \cdot x_{m,i,a,j,b} \rightarrow \min \quad (3.4.14)$$

Данная модель должна учитывать результаты решения предыдущих задач таким образом, чтобы количество посещенных POI было по меньшей мере равно $\lfloor P^*(1 - \varepsilon_P) \rfloor$, а затраченное время не превысит $[D^*(1 + \varepsilon_D)]$. Для этого к условиям (3.4.1) – (3.4.11), (3.4.13) добавим следующее:

$$\sum_{\substack{(m,i,a,j^*,t^*) \in A \\ m \neq r}} t^* \cdot x_{m,i,a,j^*,t^*} + \sum_{(r,j^*,a,j^*,t^*) \in A} a \cdot x_{r,j^*,a,j^*,t^*} \leq [D^*(1 + \varepsilon_D)] \quad (3.4.15)$$

где ε_D – заданное относительное отклонение времени прибытия от оптимального значения, полученного при решении предыдущей задачи.

Значение оптимального решения данной задачи обозначим через C^* .

3.4.4 Минимизация количества пересадок

Целевая функция для данной задачи будет иметь вид:

$$\sum_{(t,i,a,i,b) \in T} x_{t,i,a,i,b} \rightarrow \min \quad (3.4.16)$$

По аналогии с предыдущими моделями к условиям (3.4.1) – (3.4.11), (3.4.13), (3.4.15) добавим ограничение на стоимость путешествия:

$$\sum_{(m,i,a,j,b) \in A \setminus (T \cup V \cup R)} p_{m,i,a,j,b} \cdot x_{m,i,a,j,b} \leq [C^*(1 + \varepsilon_C)] \quad (3.4.17)$$

где ε_C – заданное относительное отклонение стоимости путешествия от оптимального значения, полученного при решении предыдущей задачи.

Значение оптимального решения данной задачи обозначим через H^* .

3.4.5 Минимизация общего времени пересадок и ожидания

Целевая функция для данной задачи будет иметь вид:

$$\sum_{(t,i,a,i,b) \in T} (b - a) \cdot x_{t,i,a,i,b} \rightarrow \min \quad (3.4.18)$$

К условиям (3.4.1) – (3.4.11), (3.4.13), (3.4.15), (3.4.17) добавим ограничение на количество пересадок:

$$\sum_{(t,i,a,i,b) \in T} x_{t,i,a,j,b} \leq [H^*(1 + \varepsilon_H)] \quad (3.4.19)$$

где ε_H – заданное относительное отклонение количества пересадок от оптимального значения, полученного при решении предыдущей задачи.

Значение оптимального решения данной задачи обозначим через T^* .

3.4.6 Минимизация общего времени в пешем модуле

Целевая функция для данной задачи будет иметь вид:

$$\sum_{(m,i,a,j,b) \in W} (b - a) \cdot x_{m,i,a,j,b} \rightarrow \min \quad (3.4.20)$$

К условиям (3.4.1) – (3.4.11), (3.4.13), (3.4.15), (3.4.17), (3.4.19) добавим ограничение на общее время пересадок и ожидания:

$$\sum_{(t,i,a,i,b) \in T} (b - a) \cdot x_{t,i,a,i,b} \leq [T^*(1 + \varepsilon_T)] \quad (3.4.21)$$

где ε_T – заданное относительное отклонение общего времени пересадок и ожидания от оптимального значения, полученного при решении предыдущей задачи.

Значение оптимального решения данной задачи обозначим через W^* .

3.4.7 Минимизация выброса CO₂

Целевая функция для данной задачи будет иметь вид:

$$\sum_{(m,i,a,j,b) \in A \setminus (B \cup W \cup T \cup V \cup R)} c_{m,i,a,j,b} \cdot x_{m,i,a,j,b} \rightarrow \min \quad (3.4.22)$$

К условиям (3.4.1) – (3.4.11), (3.4.13), (3.4.15), (3.4.17), (3.4.19), (3.4.21) добавим ограничение на время в пешем модуле:

$$\sum_{(m,i,a,j,b) \in W} (b - a) \cdot x_{m,i,a,j,b} \leq [W^*(1 + \varepsilon_W)] \quad (3.4.23)$$

где ε_W – заданное относительное отклонение времени в пешем модуле от оптимального значения, полученного при решении предыдущей задачи.

ГЛАВА 4

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТРОЕНИЯ МАРШРУТОВ

В данной главе описывается разработка и тестирование приложения на основании описанных выше модели и алгоритма. Приложение позволяет строить многомодальные многокритериальные туристические маршруты.

Планировщик туристических маршрутов предполагается реализовать в виде веб-сервиса.

4.1 Входные параметры

Пользовательский интерфейс должен предоставлять возможность задания критериев желаемого маршрута.

Обязательными начальными параметрами являются:

1. область построения маршрута, содержащая множество географических точек L ,
2. $i^* \in L$ – точка отправления,
3. $j^* \in L$ – точка прибытия,
4. s^* – наиболее раннее допустимое время отправления,
5. t^* – наиболее позднее допустимое время прибытия.

При планировке маршрута пользователь также может задать следующие параметры:

6. I – множество точек интереса, которые предполагается посетить,
7. d_i – длительность посещения точек, I
8. M – множество допустимых транспортных модулей,
9. v_w – средняя скорость перемещения пешком,
10. v_b – средняя скорость перемещения на велосипеде,

Средняя скорость перемещения пешком и на велосипеде задается только в случае выбора соответствующих транспортных модулей пользователем.

Пользователь также будет иметь возможность оптимизировать маршрут по следующим критериям, выбирая и ранжируя их по степени важности:

1. максимизация количества посещенных точек интереса (POI),
2. минимизация времени прибытия в конечную точку маршрута,
3. минимизация стоимости путешествия,
4. минимизация количества изменений транспортных модулей,
5. минимизация общего времени пересадок и ожидания,
6. минимизация общего времени в пешем модуль,
7. минимизация выброса CO_2 .

4.2 Архитектура приложения

Приложение состоит из трех основных модулей:

- модуль, представляющий веб-интерфейс в виде REST-сервиса
- модуль построения графа,
- модуль построения и решения задач оптимизации.

Первый модуль был реализован на языке Java с использованием фреймворков Spring Boot, Spring MVC.

Второй модуль отвечает за построение графа на основании параметров, заданных пользователем. Данный модуль использует необходимые для этого данные (например, расписание транспорта), которые предварительно сохранены в NoSQL базе данных MongoDB.

Построение и решение задач оптимизации в третьем модуле осуществляется с помощью библиотеки CPLEX.

Также был разработан вспомогательный модуль, который осуществляет сбор данных о расписании транспорта в городе Минске.

4.3 Средства разработки

4.3.1 Java

Java – сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems. Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, с помощью виртуальной Java-машины.

4.3.2 CPLEX

CPLEX – пакет программного обеспечения («решатель»), предназначенный для решения задач линейного и квадратичного программирования, в том числе целочисленного программирования.

Пакет получил своё название в честь симплекс-метода, реализованного на языке программирования Си, но впоследствии в пакете реализованы различные методы оптимизации с интерфейсом на других языках программирования. Был разработан Робертом Биксби (Robert E. Bixby).

CPLEX через абстрактный слой (Concert) может использовать API языков C++, C#, и Java, а также Python через интерфейс Си. Пакет можно использовать в связке с Microsoft Excel и MATLAB. Отдельно приложение Interactive CPLEX Optimizer может быть использовано для отладки и других задач.

Список использованных источников

1. Grodzevich O., Romanko O. Normalization and other topics in multi-objective optimization. [Electronic resource] / Grodzevich O., Romanko O. Normalization and other topics in multi-objective optimization - 2007. Mode of access: <http://goo.gl/nmAb8P>. – Date of access: 24.05.2017
2. E. Pyrga, F. Schultz, D. Wagner, C. Zaroliagis, Efficient models for timetable information in public transportation systems, ACM Journal of Experimental Algorithmics 12 (2008)
3. F. H. Meng, L. Yizhi, L. H. Wai, L. H. Chuin, A multi-criteria, multi-modal passenger route advisory system, in: Proceedings of the IES-CTR International Symposium, Singapore, 1999
4. J. Marques-Silva, J. Argelich, A. Graca, I. Lynce, Boolean lexicographic optimization: algorithms and applications, Annals of Mathematics and Artificial Intelligence 62 (3-4) (2011)
5. D. Costelloe, P. Mooney, A. Winstanley, Multi-objective optimisation on transportation networks, in: Proceedings of the 4th AGILE Conference on GIScience, 2001
6. Иржавский, П. А. Теория алгоритмов: учеб. пособие // Минск: БГУ, 2013. - 159 с.
7. Java [Electronic resource] / Java – 2017. Mode of access: <https://en.wikipedia.org/wiki/Java>. – Date of access: 24.05.2017
8. CPLEX [Electronic resource] / CPLEX – 2017. Mode of access: <https://ru.wikipedia.org/wiki/CPLEX> – Date of access: 24.05.2017

ИСХОДНЫЙ КОД ПРОГРАММЫ

BaseNode.java

```
package com.akasiyanik.trip.domain.network.nodes;

import com.akasiyanik.trip.utils.TimeUtils;

import java.time.LocalTime;

/**
 * @author akasiyanik
 */
public class BaseNode {

    private final String id;

    private final int time;

    private GeoPoint geoLocation;

    public BaseNode(String id, LocalTime time) {
        this.id = id;
        this.time = TimeUtils.timeToMinutes(time);
    }

    public BaseNode(String id, int time) {
        this.id = id;
        this.time = time;
    }

    public String getId() {
        return id;
    }

    public int getTime() {
        return time;
    }

    public LocalTime getLocalTime() {
        return TimeUtils.minutesToTime(time);
    }

    public GeoPoint getGeoLocation() {
        return geoLocation;
    }

    public void setGeoLocation(GeoPoint geoLocation) {
        this.geoLocation = geoLocation;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
```

```

        if (o == null || getClass() != o.getClass()) return false;

        BaseNode baseNode = (BaseNode) o;

        return new org.apache.commons.lang3.builder.EqualsBuilder()
            .append(id, baseNode.id)
            .append(time, baseNode.time)
            .isEquals();
    }

    @Override
    public int hashCode() {
        return new org.apache.commons.lang3.builder.HashCodeBuilder(17, 37)
            .append(id)
            .append(time)
            .toHashCode();
    }
}

```

BaseArc.java

```

package com.akasiyanik.trip.domain.network.arcs;

import com.akasiyanik.trip.domain.Mode;
import com.akasiyanik.trip.domain.network.nodes.BaseNode;
import org.apache.commons.lang3.builder.EqualsBuilder;
import org.apache.commons.lang3.builder.HashCodeBuilder;

/**
 * @author akasiyanik
 *
 */
public class BaseArc {

    private final BaseNode i;

    private final BaseNode j;

    private final Mode mode;

    public BaseArc(BaseNode i, BaseNode j, Mode mode) {
        if (i.getTime() > j.getTime()) {
            throw new RuntimeException("j node time can't be after i node time");
        }
        this.i = i;
        this.j = j;
        this.mode = mode;
    }

    public BaseNode getI() {
        return i;
    }

    public BaseNode getJ() {
        return j;
    }

    public Mode getMode() {

```

```

        return mode;
    }

    public int getTime() {
        return j.getTime() - i.getTime();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;

        if (o == null || getClass() != o.getClass()) return false;

        BaseArc baseArc = (BaseArc) o;

        return new EqualsBuilder()
            .append(i, baseArc.i)
            .append(j, baseArc.j)
            .append(mode, baseArc.mode)
            .isEquals();
    }

    @Override
    public int hashCode() {
        return newHashCodeBuilder(17, 37)
            .append(i)
            .append(j)
            .append(mode)
            .toHashCode();
    }

    @Override
    public String toString() {
        return "[m=" + mode +
            ", i=" + i.getId() +
            ", a=" + i.getTime() +
            ", j=" + j.getId() +
            ", b=" + j.getTime() + "]";
    }
}

```

RouteCriteria.java

```

package com.akasiyanik.trip.domain;

/**
 * @author akasiyanik
 */
public enum RouteCriteria {

    MAX_POI,
    MIN_TIME,
    MIN_COST,
    MIN_CHANGES,
    MIN_TIME_TRANSFER,
    MIN_TIME_WALKING,
    MIN_CO2,
    ;
}

```

ProblemSolver.java

```
package com.akasiyanik.trip.cplex;

import com.akasiyanik.trip.domain.InputParameters;
import com.akasiyanik.trip.domain.RouteCriteria;
import com.akasiyanik.trip.domain.Mode;
import com.akasiyanik.trip.domain.network.arcs.BaseArc;
import com.akasiyanik.trip.domain.network.nodes.BaseNode;
import ilog.concert.*;
import ilog.cplex.IloCplex;
import org.apache.commons.lang3.tuple.Pair;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import static com.akasiyanik.trip.utils.CplexUtil.*;

/**
 * @author akasiyanik
 */
public class ProblemSolver {

    private static final Logger logger = LoggerFactory.getLogger(ProblemSolver.class);

    private InputParameters parameters;

    private List<BaseArc> arcs;

    private int[] visitArcsMask;

    private int[] minTimeMask;

    private int[] costMask;

    private int[] co2Mask;

    private int[] transferMask;

    private int[] transferTimeMask;

    private int[] walkingTimeMask;

    private Map<String, List<Integer>> visitArcsByLocation;

    private Map<BaseNode, Set<Integer>> outgoingArcs;

    private Map<BaseNode, Set<Integer>> incomingArcs;

    private Map<BaseNode, Set<Integer>> outInTransferArcs;

    private Map<BaseNode, Map<Mode, Set<Integer>>> inTransportArcsByNode;

    private Map<BaseNode, Map<Mode, Set<Integer>>> outTransportArcsByNode;
```

```

private BaseNode startI;

private BaseNode finishJ;

private Set<Integer> startArcs;

private Set<Integer> finishArcs;

private IloIntVar[] x;

private IloCplex model;

private IloIntExpr maxPoiFunction;
private IloIntExpr minTimeFunction;
private IloIntExpr minCostFunction;
private IloIntExpr minCO2Function;
private IloIntExpr minChangesFunction;
private IloIntExpr minTimeTransferFunction;
private IloIntExpr minTimeWalkingFunction;

private IloIntExpr objectiveExpression;

private IloObjective objectiveFunction;

public ProblemSolver(List<BaseArc> arcs, InputParameters parameters) {
    this.arcs = new ArrayList<>(arcs);
    this.parameters = parameters;
    build();
}

private void build() {

    startI = new BaseNode(parameters.getDeparturePointId(),
parameters.getDepartureTime());
    finishJ = new BaseNode(parameters.getArrivalPointId(),
parameters.getArrivalTime());

    outgoingArcs = getOutgoingArcsByNodes(arcs);
    incomingArcs = getIncomingArcsByNodes(arcs);

    outInTransferArcs = getInOutTransferArcsByNodes(arcs);

    inTransportArcsByNode = getInTransportArcsByNodes(arcs);
    outTransportArcsByNode = getOutTransportArcsByNodes(arcs);

    startArcs = outgoingArcs.remove(startI);
    finishArcs = incomingArcs.remove(finishJ);

    visitArcsMask = new int[arcs.size()];
    visitArcsByLocation = IntStream
        .range(0, arcs.size())
        .filter(i -> arcs.get(i).getMode().equals(Mode.VISIT))
        .peek(i -> visitArcsMask[i] = 1)
        .boxed()
        .collect(Collectors.groupingBy(i -> arcs.get(i).getId()));
}

```

```

minTimeMask = new int[arcs.size()];
finishArcs.forEach(i -> {
    BaseArc arc = arcs.get(i);
    if (arc.getMode() == Mode.DUMMY_START_FINISH) {
        minTimeMask[i] = arc.getI().getTime();
    } else {
        minTimeMask[i] = arc.getJ().getTime();
    }
});

costMask = new int[arcs.size()];
IntStream
    .range(0, arcs.size())
    .filter(i -> arcs.get(i).getMode().isTransport())
    .forEach(i -> costMask[i] = arcs.get(i).getMode().getCost());

co2Mask = new int[arcs.size()];
IntStream
    .range(0, arcs.size())
    .filter(i -> arcs.get(i).getMode().isCO2Transport())
    .forEach(i -> {
        BaseArc arc = arcs.get(i);
        co2Mask[i] = arc.getMode().getCo2() * arc.getTime();
    });

transferMask = new int[arcs.size()];
transferTimeMask = new int[arcs.size()];
IntStream
    .range(0, arcs.size())
    .filter(i -> arcs.get(i).getMode().equals(Mode.TRANSFER))
    .forEach(i -> {
        transferMask[i] = 1;
        transferTimeMask[i] = arcs.get(i).getTime();
    });

walkingTimeMask = new int[arcs.size()];
IntStream
    .range(0, arcs.size())
    .filter(i -> arcs.get(i).getMode().equals(Mode.WALK))
    .forEach(i -> {
        walkingTimeMask[i] = arcs.get(i).getTime();
    });

}

public List<BaseArc> solve() {

    try {
        model = new IloCplex();
        x = model.boolVarArray(arcs.size());

        addMandatoryConstraints();

        List<BaseArc> result = null;
        List<Pair<RouteCriteria, Double>> criteria = parameters.getCriteria();

        double objectiveValue = 0.0;

        for (int i = 0; i < criteria.size(); i++) {

```

```

        if (i > 0) {
            Pair<RouteCriteria, Double> prevCriteria = criteria.get(i - 1);
            addConstraintFromPreviousProblem(prevCriteria.getLeft(),
            prevCriteria.getRight(), objectiveValue);
        }
        addObjectiveFunction(criteria.get(i).getLeft());

        model.exportModel("trip" + i + ".lp");

        logger.info("CPLEX problem solving...");

        boolean isSolved = model.solve();

        if (isSolved) {
            logger.info("CPLEX Solution status = " + model.getStatus());
            objectiveValue = model.getObjValue();
            logger.info("Solution value = " + objectiveValue);

            result = new ArrayList<>();
            double[] values = model.getValues(x);
            for (int j = 0; j < x.length; ++j) {
                logger.debug("Variable " + j + ": Value = " + values[j]);
                if (values[j] == 1) {
                    result.add(arcs.get(j));
                }
            }
            Collections.sort(result, (a1, a2) -> a1.getI().getTime() -
a2.getI().getTime());
        } else {
            logger.warn("CPLEX Solution status = " + model.getStatus());
        }
    }

    return result;
} catch (IloException e) {
    throw new RuntimeException(e);
}
}

private void addMandatoryConstraints() throws IloException {

    //constraints (3) - (4)
    addUniqueInOutArcsConstraint(model, x, incomingArcs);
    addUniqueInOutArcsConstraint(model, x, outgoingArcs);

    //constraints (1) - (2)
    addOnlyInOutArcConstraint(model, x, startArcs);
    addOnlyInOutArcConstraint(model, x, finishArcs);

    // constraint (5)
    addEqualInOutForIntermediateNodesConstraint(model, x, outgoingArcs, incomingArcs);

    //constraint(6)
    addAtMostOneTransferArcForNodeConstraint(model, x, outInTransferArcs);

    //constraint (7)
    addAtMostOneVisitArcForLocationConstraint(model, x, visitArcsByLocation);
}

```

```

//constraint (8)
addStartOnlyInSpecifiedLocationConstraint(model, x, outgoingArcs, startI);

//constraint(9)
addRequiredTransferBetweenTransportModesConstraint(model, x,
inTransportArcsByNode, outTransportArcsByNode);

}

private void addConstraintFromPreviousProblem(RouteCriteria criteria, double coeff,
double prevObjectiveResult) throws IloException {

    switch (criteria) {
        case MAX_POI: {
            model.addGe(objectiveExpression, Math.floor(prevObjectiveResult * (1 -
coeff)));
            break;
        }
        default: {
            model.addLe(objectiveExpression, Math.ceil(prevObjectiveResult * (1 +
coeff)));
        }
    }
}

private void addObjectiveFunction(RouteCriteria criteria) throws IloException {
    if (objectiveFunction != null) {
        model.remove(objectiveFunction);
    }

    switch (criteria) {
        case MAX_POI: {
            objectiveExpression = getMaxPoiFunction();
            objectiveFunction = model.addMaximize(objectiveExpression);
            break;
        }
        case MIN_TIME: {
            objectiveExpression = getMinTimeFunction();
            objectiveFunction = model.addMinimize(objectiveExpression);
            break;
        }
        case MIN_COST: {
            objectiveExpression = getMinCostFunction();
            objectiveFunction = model.addMinimize(objectiveExpression);
            break;
        }
        case MIN_CHANGES: {
            objectiveExpression = getMinChangesFunction();
            objectiveFunction = model.addMinimize(objectiveExpression);
            break;
        }
        case MIN_TIME_TRANSFER: {
            objectiveExpression = getMinTimeTransferFunction();
            objectiveFunction = model.addMinimize(objectiveExpression);
            break;
        }
        case MIN_TIME_WALKING: {
            objectiveExpression = getMinTimeWalkingFunction();
            objectiveFunction = model.addMinimize(objectiveExpression);
            break;
        }
    }
}

```

```

        break;
    }
    case MIN_C02: {
        objectiveExpression = getMinC02Function();
        objectiveFunction = model.addMinimize(objectiveExpression);
        break;
    }
}

private IloIntExpr getMaxPoiFunction() throws IloException {
    if (maxPoiFunction == null) {
        maxPoiFunction = model.scalProd(visitArcsMask, x);
    }
    return maxPoiFunction;
}

private IloIntExpr getMinTimeFunction() throws IloException {
    if (minTimeFunction == null) {
        minTimeFunction = model.scalProd(minTimeMask, x);
    }
    return minTimeFunction;
}

private IloIntExpr getMinCostFunction() throws IloException {
    if (minCostFunction == null) {
        minCostFunction = model.scalProd(costMask, x);
    }
    return minCostFunction;
}

private IloIntExpr getMinC02Function() throws IloException {
    if (minC02Function == null) {
        minC02Function = model.scalProd(co2Mask, x);
    }
    return minC02Function;
}

private IloIntExpr getMinChangesFunction() throws IloException {
    if (minChangesFunction == null) {
        minChangesFunction = model.scalProd(transferMask, x);
    }
    return minChangesFunction;
}

private IloIntExpr getMinTimeTransferFunction() throws IloException {
    if (minTimeTransferFunction == null) {
        minTimeTransferFunction = model.scalProd(transferTimeMask, x);
    }
    return minTimeTransferFunction;
}

private IloIntExpr getMinTimeWalkingFunction() throws IloException {
    if (minTimeWalkingFunction == null) {
        minTimeWalkingFunction = model.scalProd(walkingTimeMask, x);
    }
    return minTimeWalkingFunction;
}

```

```
}
```

CPLEXUtils.java

```
package com.akasiyanik.trip.utils;

import com.akasiyanik.trip.domain.Mode;
import com.akasiyanik.trip.domain.network.arcs.BaseArc;
import com.akasiyanik.trip.domain.network.nodes.BaseNode;
import ilog.concert.IloException;
import ilog.concert.IloIntVar;
import ilog.concert.IloNumExpr;
import ilog.concert.IloNumVar;
import ilog.cplex.IloCplex;

import java.util.*;

/**
 * @author akasiyanik
 */
public final class CplexUtil {

    public static Map<BaseNode, Set<Integer>> getOutgoingArcsByNodes(List<BaseArc> arcs) {
        Map<BaseNode, Set<Integer>> outgoingArcs = new HashMap<>();
        int index = 0;
        for (BaseArc arc : arcs) {
            // out
            BaseNode arcI = arc.getI();
            Set<Integer> indexesOut = outgoingArcs.get(arcI);
            if (indexesOut == null) {
                indexesOut = new HashSet<>();
                outgoingArcs.put(arcI, indexesOut);
            }
            indexesOut.add(index);
            index++;
        }
        return outgoingArcs;
    }

    public static Map<BaseNode, Set<Integer>> getIncomingArcsByNodes(List<BaseArc> arcs) {
        Map<BaseNode, Set<Integer>> incomingArcs = new HashMap<>();
        int index = 0;
        for (BaseArc arc : arcs) {
            // in
            BaseNode arcJ = arc.getJ();
            Set<Integer> indexesIn = incomingArcs.get(arcJ);
            if (indexesIn == null) {
                indexesIn = new HashSet<>();
                incomingArcs.put(arcJ, indexesIn);
            }
            indexesIn.add(index);
            index++;
        }
        return incomingArcs;
    }

    public static Map<BaseNode, Set<Integer>> getInOutTransferArcsByNodes(List<BaseArc>
```

```

allArcs) {
    Map<BaseNode, Set<Integer>> result = new HashMap<>();
    int index = 0;
    for (BaseArc arc : allArcs) {
        if (arc.getMode().equals(Mode.TRANSFER)) {
            // in
            BaseNode arcI = arc.getI();
            Set<Integer> indexesI = result.get(arcI);
            if (indexesI == null) {
                indexesI = new HashSet<>();
                result.put(arcI, indexesI);
            }
            indexesI.add(index);

            BaseNode arcJ = arc.getJ();
            Set<Integer> indexesJ = result.get(arcJ);
            if (indexesJ == null) {
                indexesJ = new HashSet<>();
                result.put(arcJ, indexesJ);
            }
            indexesJ.add(index);

            index++;
        }
    }
    return result;
}

public static Map<BaseNode, Map<Mode, Set<Integer>>>
getInTransportArcsByNodes(List<BaseArc> allArcs) {
    Map<BaseNode, Map<Mode, Set<Integer>>> result = new HashMap<>();
    int index = 0;
    for (BaseArc arc : allArcs) {
        Mode mode = arc.getMode();
        if (Mode.TRANSPORT.contains(mode)) {

            // in
            BaseNode arcJ = arc.getJ();
            Map<Mode, Set<Integer>> indexesByModes = result.get(arcJ);
            if (indexesByModes == null) {
                indexesByModes = new HashMap<>();
                result.put(arcJ, indexesByModes);
            }
            Set<Integer> indexesJ = indexesByModes.get(mode);
            if (indexesJ == null) {
                indexesJ = new HashSet<>();
                indexesByModes.put(mode, indexesJ);
            }
            indexesJ.add(index);

            index++;
        }
    }
    return result;
}

public static Map<BaseNode, Map<Mode, Set<Integer>>>
getOutTransportArcsByNodes(List<BaseArc> allArcs) {
    Map<BaseNode, Map<Mode, Set<Integer>>> result = new HashMap<>();
    int index = 0;

```

```

        for (BaseArc arc : allArcs) {
            Mode mode = arc.getMode();
            if (Mode.TRANSPORT.contains(mode)) {

                // out
                BaseNode arcI = arc.getI();
                Map<Mode, Set<Integer>> indexesByModes = result.get(arcI);
                if (indexesByModes == null) {
                    indexesByModes = new HashMap<>();
                    result.put(arcI, indexesByModes);
                }
                Set<Integer> indexesI = indexesByModes.get(mode);
                if (indexesI == null) {
                    indexesI = new HashSet<>();
                    indexesByModes.put(mode, indexesI);
                }
                indexesI.add(index);
                index++;
            }
        }
        return result;
    }

    public static void addRequiredTransferBetweenTransportModesConstraint(IloCplex model,
    IloIntVar[] x, Map<BaseNode, Map<Mode, Set<Integer>>> inTransportArcs, Map<BaseNode,
    Map<Mode, Set<Integer>>> outTransportArcs) throws IloException {
        Set<BaseNode> nodes = new HashSet<>();
        nodes.addAll(inTransportArcs.keySet());
        nodes.addAll(outTransportArcs.keySet());

        for (BaseNode node : nodes) {

            Map<Mode, Set<Integer>> inArcsByMode = inTransportArcs.get(node);
            Map<Mode, Set<Integer>> outArcsByMode = outTransportArcs.get(node);

            if (inArcsByMode != null && outArcsByMode != null) {

                Set<Mode> outModes = outArcsByMode.keySet();
                Map<Mode, IloNumExpr> outArcsSums = new HashMap<>();
                for (Mode outMode : outModes) {
                    Set<Integer> outArcs = outArcsByMode.get(outMode);
                    IloNumVar[] outArcsVariables = outArcs
                        .stream()
                        .map(ind -> x[ind])
                        .toArray(IloNumVar[]::new);
                    IloNumExpr outSum = model.sum(outArcsVariables);
                    outArcsSums.put(outMode, outSum);
                }
            }

            for (Mode inMode : inArcsByMode.keySet()) {

                Set<Integer> inArcs = inArcsByMode.get(inMode);

                IloNumVar[] inArcsVariables = inArcs
                    .stream()
                    .map(ind -> x[ind])
                    .toArray(IloNumVar[]::new);
            }
        }
    }
}

```

```

        IloNumExpr inArcsSum = model.sum(inArcsVariables);

        for (Mode outMode : outModes) {
            if (!inMode.equals(outMode)) {
                model.addLe(model.sum(inArcsSum, outArcsSums.get(outMode)),
1.0);
            }
        }
    }

}

public static void addStartOnlyInSpecifiedLocationConstraint(IloCplex model,
IloIntVar[] x, Map<BaseNode, Set<Integer>> outgoingArcs, BaseNode startI) throws
IloException {
    int startTime = startI.getTime();
    for (Map.Entry<BaseNode, Set<Integer>> nodeWithArcs : outgoingArcs.entrySet()) {
        if (nodeWithArcs.getKey().getTime() == startTime) {
            IloNumVar[] arcsVariables = nodeWithArcs.getValue()
                .stream()
                .map(ind -> x[ind])
                .toArray(IloNumVar[]::new);
            model.addEq(model.sum(arcsVariables), 0.0);
        }
    }
}

public static void addAtMostOneVisitArcForLocationConstraint(IloCplex model,
IloIntVar[] x, Map<String, List<Integer>> visitingArcs) throws IloException {
    for (List<Integer> arcsPerLocation : visitingArcs.values()) {
        if (arcsPerLocation.size() > 1) {
            IloNumVar[] arcsVariables = arcsPerLocation
                .stream()
                .map(ind -> x[ind])
                .toArray(IloNumVar[]::new);
            model.addLe(model.sum(arcsVariables), 1.0);
        }
    }
}

public static void addAtMostOneTransferArcForNodeConstraint(IloCplex model,
IloIntVar[] x, Map<BaseNode, Set<Integer>> transferArcsByNode) throws IloException {
    for (Set<Integer> transferArcs : transferArcsByNode.values()) {
        if (transferArcs.size() > 1) {
            IloNumVar[] arcsVariables = transferArcs
                .stream()
                .map(ind -> x[ind])
                .toArray(IloNumVar[]::new);
            model.addLe(model.sum(arcsVariables), 1.0);
        }
    }
}

public static void addEqualInOutForIntermediateNodesConstraint(IloCplex model,
IloIntVar[] x, Map<BaseNode, Set<Integer>> outgoingArcs, Map<BaseNode, Set<Integer>>

```

```

incomingArcs) throws IloException {
    Set<BaseNode> nodes = new HashSet<>();
    nodes.addAll(outgoingArcs.keySet());
    nodes.addAll(incomingArcs.keySet());

    for (BaseNode node : nodes) {
        Set<Integer> out = outgoingArcs.get(node);
        Set<Integer> in = incomingArcs.get(node);

        IloNumVar[] inArcsVariables = null;
        if (in != null) {
            inArcsVariables = in
                .stream()
                .map(ind -> x[ind])
                .toArray(IloNumVar[]::new);
        }

        IloNumVar[] outArcsVariables = null;
        if (out != null) {
            outArcsVariables = out
                .stream()
                .map(ind -> x[ind])
                .toArray(IloNumVar[]::new);
        }

        if (inArcsVariables != null && outArcsVariables != null) {
            model.addEq(model.sum(inArcsVariables), model.sum(outArcsVariables));
        } else if (inArcsVariables != null && outArcsVariables == null) {
            model.addEq(model.sum(inArcsVariables), 0);
        } else if (inArcsVariables == null && outArcsVariables != null) {
            model.addEq(model.sum(outArcsVariables), 0);
        } else {
            continue;
        }
    }
}

public static void addOnlyInOutArcConstraint(IloCplex model, IloIntVar[] x,
Set<Integer> arcs) throws IloException {
    IloNumVar[] arcsVariables = arcs
        .stream()
        .map(ind -> x[ind])
        .toArray(IloNumVar[]::new);
    model.addEq(model.sum(arcsVariables), 1.0);
}

public static void addUniqueInOutArcsConstraint(IloCplex model, IloIntVar[] x,
Map<BaseNode, Set<Integer>> indexesByNode) throws IloException {
    for (Set<Integer> indexes : indexesByNode.values()) {
        if (indexes.size() > 1) {
            IloNumVar[] arcsVariables = indexes.stream().map(ind ->
x[ind]).toArray(IloNumVar[]::new);
            model.addLe(model.sum(arcsVariables), 1.0);
        }
    }
}
}

```