**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, 100 с., 29 рис., 30 источников, 1 приложение.

ЦИФРОВОЕ ИЗОБРАЖЕНИЕ, ОСОБЫЕ ТОЧКИ, ДЕТЕКТОРЫ, ДЕСКРИПТОРЫ, МАШИННОЕ ОБУЧЕНИЕ, КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ, BAG-OF-WORDS, АЛГОРИТМ.

Объектом исследования цифровые изображения монет и банкнот.

Целью работы является изучение алгоритмов детектирования особых точек изображения, построения дескрипторов особых точек и изображений, машинного обучения и классификации изображений. А также разработка приложения, позволяющего классифицировать изображения различных объектов (монет, банкнот).

В результате работы было разработано приложение для классификации объектов двух классов, обучающееся на тренировочной выборке по алгоритму bag-of-words.

Методы исследования: изучение литературы, классификация изображений, теория алгоритмов, машинное обучение.

Область применения: создание приложения, распознающего монеты, для мобильных платформ.

**Рэферат**

Дыпломная праца, 100 с., 29 мал., 30 крыніц, 1 дадатак.

ЛІЧБАВЫ МАЛЮНАК, АСОБЫЯ КРОПКІ, ДЭТЭКТАРЫ, ДЗЕСКРЫПТАРЫ, МАШЫННАЕ НАВУЧАННЕ, КЛАСІФІКАЦЫЯ МАЛЮНКАЎ, BAG-OF-WORDS, АЛГАРЫТМ

Аб’ектам даследавання з’яўляюцца лічбавыя малюнкі манет і банкнот.

Мэтай працы з’яўляецца вывучэнне алгарытмаў дэтэктавання асобых кропак малюнка, пастраенне дзескрыптараў асобых кропак і малюнкаў, машыннага навучання і класіфікацыі малюнкаў. А таксама распрацоўка праграмы, якая дазваляе класіфікаваць малюнкі розных аб’ектаў (манет, банкнот).

У выніку працы была распрацавана праграма для класіфікацыі аб’ектаў двух класаў, якая навучаецца на трэніровачнай выбарцы па алгарытму bag-of-words.

Метады даследавання: вывучэнне літаратуры, класіфікацыя малюнкаў, тэорыя алгарытмаў, машыннае навучанне.

Вобласць выкарыстання: распрацоўка праграмы, якая распазнае манеты, для мабільных платформ.

**Abstract**

Graduation work, 100 pp., 29 pictures, 30 sources, 1 appendix.

DIGITAL IMAGES, FEATURES, DETECTORS, DESCRIPTORS, MACHINE LEARNING, IMAGE CLASSIFICATION, BAG-OF-WORDS, ALGORITHMS.

Object of research: digital images of coins and banknotes.

Goal of the research: studying algorithms of image features detection, building descriptors of features and images, machine learning and image classification. And the development of the application for classification images of different objects (coins, banknotes).

The result of current research is the application for classification images of 2 types, working on the base of bag-of-words algorithm.

Research methods: analysis of topic relevant literature, image classification, algorithms theory, machine learning.

Applications: development of a mobile-platform application for coins recognition.

**ОГЛАВЛЕНИЕ**

ВВЕДЕНИЕ 6

ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ 7

ГЛАВА 2 ПОДХОДЫ К РЕШЕНИЮ 8

2.1 «Развернутый во времени» и «изменяющийся во времени» графы 8

2.1 Постановка задачи 9

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 Моделирование «изменяющегося во времени» графа 16

3.2 Точки интереса 18

3.2 Формулирование задач булевского программирования 19

3.3.1 Максимизация количества посещенных точек интереса 19

2.3.2 Минимизация времени прибытия в конечную точку маршрута 21

2.3.3 Минимизация стоимости путешествия 21

2.3.4 Минимизация количества пересадок 22

2.3.5 Минимизация общего времени пересадок и ожидания 22

2.3.4 Минимизация общего времени в пешем модуле 23

2.3.4 Минимизация выброса 23

ГЛАВА 4 РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТРОЕНИЯ ТУРИСТИЧЕСКИХ МАРШРУТОВ 26

4.1 Постановка задачи 26

4.2 Средства разработки 28

4.3 Вычислительные эксперименты и их результаты 29

ПРИЛОЖЕНИЕ А 30

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 многомодальные варианты, оптимизированные 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 алгоритма создания многомодальных многокритериальных маршрутов для городского туризма.

ГЛАВА 1   
ПОСТАНОВКА ЗАДАЧИ

ГЛАВА 2   
ПОДХОДЫ К РЕШЕНИЮ

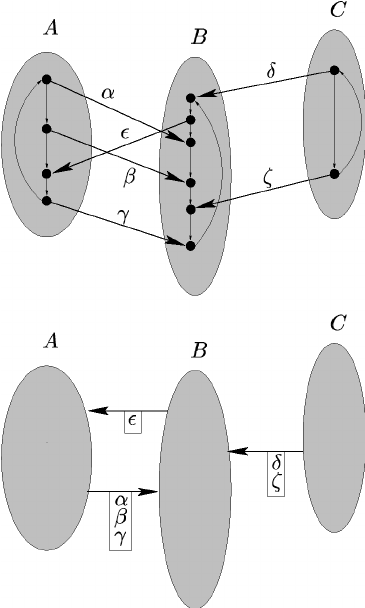
2.1 «Развернутый во времени» и «изменяющийся во времени» графы

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.

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

2.1 Постановка задачи

Многокритериальная задача оптимизации может быть записана в следующем виде:

|  |  |
| --- | --- |
| , | (1.1.1) |

где  - целевые функции, которые могут конфликтовать друг с другом, и - область допустимых решений.

Задачи максимизации можно привести к задачам минимизации следующим образом:

|  |  |
| --- | --- |
| , | (1.1.2) |

Цель многокритериальной задачи оптимизации – одновременно минимизировать все целевые функции.

Обозначим за отображение пространства решений в область допустимых целевых значений, т.е. каждому элементу пространства решений сопоставим вектор из значений целевых функций на этом элементе:

|  |  |
| --- | --- |
|  | (1.1.3) |

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

|  |  |
| --- | --- |
|  | (1.2.4) |

Обозначим *D1* множество ее оптимальных решений. Если *D1* – одноэлементное множество, то единственное оптимальное решение задачи 12.4) является одновременно оптимальным по принципу ЛУК исходной многокритериальной задачи (1.1.1). В противном случае далее решаем задачу

|  |  |
| --- | --- |
|  | (1.2.5) |

Обозначим D2 множество оптимальных решений задачи (1.2.5). Если *D2* – одноэлементное множество, то единственное оптимальное решение задачи (1.2.5) является одновременно оптимальным по принципу лексикографического упорядочения критериев решением исходной многокритериальной задачи (1.1.1). В противном случае поступаем аналогично предыдущему – решаем задачу минимизации значения критерия *Q3(x)* при условии, что *х ∈ D2*, и т.д. Максимальное число последовательно решаемых однокритериальных задач равно m, т.е. числу критериев исходной многокритериальной задачи. Если решение задачи (1.1.1) определилось в результате выполнения меньшего числа этапов, мы остановились на *k*-ой итерации, то оно единственно. 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

|  |  |
| --- | --- |
|  | (1.2.6) |

в которой критерий *Q1(x)* считаем ведущим. Реализуя этот метод, сначала находим решение *х\*,* оптимальное при лексикографическом упорядочении критериев *(Q1(x), Q2(x)).* Пусть *(Q1(x\*), Q2(x\*)) = (a, b).* Точка *(a, b)* – оценка найденного решения. Если данная оценка (первая ее координата *a* – максимально возможное значение критерия Q1(x), а вторая координата *b* – максимально возможное значение критерия *Q2(x)*) удовлетворяет лицо, принимающее решение, то *х\** принимается за искомое оптимально-компромиссное решение; в противном случае ЛПР назначает уступку *δ1, δ1 > 0* по допустимому значению первого критерия. Далее решается задача

|  |  |
| --- | --- |
|  | (1.2.7) |

при дополнительном условии

|  |  |
| --- | --- |
| *Q*1(*x*) ≥ *a* – δ1*.* | (1.2.8) |

Пусть оптимальное решение х\*\* задачи в исходной задаче (1.2.6) имеет оценку *(a1, b1).* Очевидно, *b1 ≥ b*. Если данная оценка удовлетворяет ЛПР, то *х\*\** принимается за искомое оптимально-компромиссное решение задачи (1.2.6) в противном случае ЛПР назначает новую уступку *δ2*, где *δ2 > δ1*, по допустимому значению первого критерия. Далее решается задача

|  |  |
| --- | --- |
|  | (1.2.9) |

при дополнительном условии

|  |  |
| --- | --- |
| *Q1(x) ≥ a – δ2.* | (1.2.10) |

Пусть оптимальное решение *х\*\*\** задачи (1.2.9) - (1.2.10) в исходной задаче (1.2.6) имеет оценку *(a2, b2).* Как очевидно, *b2 ≥ b1*. Если данная оценка удовлетворяет ЛПР, то *х\*\*\** принимается за искомое оптимально-компромиссное решение задачи (1.2.6); в противном случае ЛПР назначает новую уступку *δ3, δ3 > δ2*, 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  - главный критерий. Тогда задача (1.1.1) сводится к однокритериальной задаче

|  |  |
| --- | --- |
|  | (2.2.11) |

при дополнительных условиях

|  |  |
| --- | --- |
| , *k* = 2, 3, …, *m*; | (2.2.12) |

где *hk* – 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 *Qi*, 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

|  |  |
| --- | --- |
|  | (2.2.13) |

при условии х ∈ D.

При этом коэффициенты имеют следующие ограничения:

|  |  |
| --- | --- |
| , , | (2.2.14) |
|  | (2.2.15) |

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 нормализирующей 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-фронта.

Можно заметить, что размеры множества Парето-оптимальных альтернатив предоставляют информацию для решения задачи. Компоненты  идеального целевого вектора определяются минимизацией каждой целевой функции независимо от остальных, т. е.:

Идеальный вектор  называемый точкой Утопия, не всегда может являться допустимым, так как могут существовать конфликты между целевыми функциями. Данная точка дает нам информацию о нижней границе Парето-фронта.

Верхние границы Парето-фронта определяются компонентами точки Надира:



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



Можно заметить, что после нормализации целевые функции будут ограничены:



что дает одинаковый интервал для изменения целевых функций.

Для вычисления нормализационного интервала  необходимо решить k задач оптимизации вида для получения x[i]. Значение этих элементов является ключевым, так как они необходимы для вычисления  и .

ГЛАВА 3 ПОСТРОЕНИЕ МОДЕЛИ И АЛГОРИТМА РЕШЕНИЯ

3.1 Моделирование «изменяющегося во времени» графа

Обозначим через множество географических точек (локаций):

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

Построим модель взвешенной сети используя «изменяющийся во времени» граф , где – множество вершин, – множество ребер. Вершины представлены в виде пар где – географическая локация, – время события. Две вершины и соединены ребром тогда и только тогда, если турист может отправиться из точки в момент времени и прибыть в точку в момент времени , используя транспортный модуль (вид транспорта) . Таким образом, если несколько видов транспорта позволяют выехать из точки в момент и прибыть в в момент , то вершины будут соединены несколькими параллельными дугами, отличающимися в .

Разрабатываемая модель также 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 и соединены дугой ожидания

Каждая дуга будет иметь множество весов:

* – расстояние между вершинами, которое будет пройдено используя режим
* – средняя скорость для режима на данном участке. Для транспорта с фиксированным расписанием будем полагать равной рекомендованной скорости. Для пешего и велосипедного режима , где – стандартная средняя скорость, – высокая средняя скорость.
* – cтоимость перемещения при использовании модуля на данном участке ( – для пешего и велосипедного режима и режима ожидания).
* – количество выбрасываемого на человека ( – для пешего, велосипедного модуля и модуля ожидания).

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

Для транспорта без фиксированного расписания будем полагать, что такая дуга будет существовать тогда и только тогда, если и перемещение из точки в точку во временном интервале 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 Точки интереса

*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 – множество точек интереса и . Каждой точке присвоим временной интервал , который определяет время, когда турист может посетить точку . Например, интервал может соответствовать времени работы музея*. Продолжительность посещения*  точки интереса будет задаваться пользователем.

Чтобы смоделировать возможность посещения туристом POI, введем дополнительный модуль – *модуль посещения* и положим, что вершины и соединены *дугой посещения* тогда и только тогда, когда , , и .

Стоимость перемещения и количество выбросов для дуг посещения будет равно 0.

// РИСУНОК

3.2 Формулирование задач булевского программирования

В данной главе опишем формулирование задач булевского программирования для модели 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 .

Составление задач для иного порядка критериев происходит по аналогичной схеме.

3.3.1 Максимизация количества посещенных точек интереса

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

|  |  |
| --- | --- |
|  | (2.3.1) |

где

– множество точек POI, выбранных пользователем для посещения,

– множество дуг посещения.

Сформулируем условия для данной задачи.

Маршрут должен содержать в точности одну дугу, выходящую из начальной вершины :

|  |  |
| --- | --- |
|  | (2.3.2) |

Маршрут должен содержать в точности одну дугу, входящую в конечную вершину :

|  |  |
| --- | --- |
|  | (2.3.3) |

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

|  |  |
| --- | --- |
|  | (2.3.4) |
|  | (2.3.5) |

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

|  |  |
| --- | --- |
|  | (2.3.6) |

Не более одной дуги пересадки для каждой вершины пути:

Каждая точка интереса посещается не более одного раза:

|  |  |
| --- | --- |
|  | (2.3.7) |

В начальный момент времени дуги выходят только из стартовой точки :

|  |  |
| --- | --- |
|  | (2.3.8) |

Переход между модулями происходит только посредством трансферной дуги:

|  |  |
| --- | --- |
|  | (2.3.9) |
|  | (2.3.10) |

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

2.3.2 Минимизация времени прибытия в конечную точку маршрута

Целевая функция для данной задачи будет иметь следующий вид:

|  |  |
| --- | --- |
|  | (2.3.11) |

В данной модели – искусственный модуль, который используется для перемещения туриста в финальной точек из любого момента времени в момент времени .

К условиям (2.3.2) – (2.3.10) добавим ограничение на количество точек для посещения так, чтобы пользователь по меньшей мере посетил точек, где – относительное отклонение количества посещенных POI от оптимума:

|  |  |
| --- | --- |
|  | (2.3.12) |

Значение оптимального решения данной задачи обозначим через .

2.3.3 Минимизация стоимости путешествия

Целевая функция для данной задачи будет иметь следующий вид:

|  |  |
| --- | --- |
|  | (2.3.13) |

Данная модель должна учитывать результаты решения предыдущих задач таким образом, чтобы количество посещенных POI было по меньшей мере равно , а затраченное время не превысит . Для этого к условиям (2.3.2) – (2.3.10), (2.3.12) добавим следующее:

|  |  |
| --- | --- |
|  | (2.3.14) |

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

Значение оптимального решения данной задачи обозначим через .

2.3.4 Минимизация количества пересадок

Целевая функция для данной задачи будет иметь вид:

|  |  |
| --- | --- |
|  | (2.3.15) |

По аналогии с предыдущими моделями к условиям (2.3.2) – (2.3.10), (2.3.12), (2.3.14) добавим ограничение на стоимость путешествия:

|  |  |
| --- | --- |
|  | (2.3.16) |

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

Значение оптимального решения данной задачи обозначим через .

2.3.5 Минимизация общего времени пересадок и ожидания

Целевая функция для данной задачи будет иметь вид:

|  |  |
| --- | --- |
|  | (2.3.17) |

К условиям (2.3.2) – (2.3.10), (2.3.12), (2.3.14), (2.3.16) добавим ограничение на количество пересадок:

|  |  |
| --- | --- |
|  | (2.3.18) |

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

Значение оптимального решения данной задачи обозначим через .

2.3.4 Минимизация общего времени в пешем модуле

Целевая функция для данной задачи будет иметь вид:

|  |  |
| --- | --- |
|  | (2.3.19) |

К условиям (2.3.2) – (2.3.10), (2.3.12), (2.3.14), (2.3.16), (2.3.18) добавим ограничение на общее время пересадок и ожидания:

|  |  |
| --- | --- |
|  | (2.3.20) |

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

Значение оптимального решения данной задачи обозначим через .

2.3.4 Минимизация выброса

Целевая функция для данной задачи будет иметь вид:

|  |  |
| --- | --- |
|  | (2.3.21) |

К условиям (2.3.2) – (2.3.10), (2.3.12), (2.3.14), (2.3.16), (2.3.18), (2.3.20) добавим ограничение на время в пешем модуле:

|  |  |
| --- | --- |
|  | (2.3.21) |

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

ГЛАВА 4  
РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ПОСТРОЕНИЯ ТУРИСТИЧЕСКИХ МАРШРУТОВ

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

4.1 Постановка задачи

Планировщик туристических маршрутов предполагается реализовать в виде веб-сервиса. Пользовательский интерфейс должен предоставлять возможность задания критериев желаемого маршрута.

Обязательными начальными параметрами являются:

1. область построения маршрута, содержащая множество географических точек ,
2. - точка отправления,
3. - точка прибытия,
4. - наиболее раннее допустимое время отправления,
5. - наиболее позднее допустимое время прибытия.

При планировке маршрута пользователь также может задать следующие параметры:

1. – множество точек интереса, которые предполагается посетить,
2. – длительность посещения точек,
3. – множество допустимых транспортных модулей,
4. – средняя скорость перемещения пешком,
5. – 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 .

4.2 Средства разработки

Для анализа исходных данных, а также разработки последующих моделей планируется использование следующих технологий и инструментов обработки больших объемов информации:

* Apache Spark (в том числе GraphX)
* Язык программирования Java 1.8

*Apache Spark*— программный каркас с открытым исходным кодом для реализации распределённой обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop. В отличие от классического обработчика из ядра Hadoop, реализующего двухуровневую концепцию MapReduce с дисковым хранилищем, использует специализированные примитивы для рекурентной обработки в оперативной памяти, благодаря чему позволяет получать значительный выигрыш в скорости работы для некоторых классов задач, в частности, возможность многократного доступа к загруженным в память пользовательским данным делает библиотеку привлекательной для алгоритмов машинного обучения.

Проект предоставляет программные интерфейсы для языков Java, Scala, Python, R. Состоит из ядра и нескольких расширений, таких как Spark SQL (позволяет выполнять SQL-запросы над данными), Spark Streaming (надстройка для обработки потоковых данных), Spark MLlib (набор библиотек машинного обучения), GraphX (предназначено для распределённой обработки графов). Может работать как в среде кластера Hadoop под управлением YARN, так и без компонентов ядра Hadoop, поддерживает несколько распределённых систем хранения — HDFS, OpenStack Swift, NoSQL-СУБД Cassandra, Amazon S3.

4.3 Вычислительные эксперименты и их результаты

Ниже представлены результаты работы программы по классификации выборки изображений

ПРИЛОЖЕНИЕ А

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 new HashCodeBuilder(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).getI().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;  
 }  
 case MIN\_CO2: {  
 objectiveExpression = getMinCO2Function();  
 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 getMinCO2Function() throws IloException {  
 if (minCO2Function == null) {  
 minCO2Function = model.scalProd(co2Mask, x);  
 }  
 return minCO2Function;  
 }  
  
 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);  
 }  
 }  
 }  
}