

Государственное образовательное учреждение
высшего профессионального образования
“Томский государственный университет”

На правах рукописи

Пожидаев Михаил Сергеевич

АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ
МАРШРУТИЗАЦИИ ТРАНСПОРТА

Специальность 05.13.18 — Математическое моделирование,
численные методы и комплексы программ

Диссертация
на соискание ученой степени
кандидата технических наук

Научный руководитель
д-р тех. наук, профессор
Костюк Юрий Леонидович

Томск — 2010

Оглавление

Введение	5
1 Обзор алгоритмов решения ЗМТ	13
1.1 Терминология	13
1.2 Постановка ЗМТ	14
1.3 Классификация алгоритмов для решения ЗМТ	17
1.4 Конструктивные классические алгоритмы	19
1.4.1 Алгоритм Кларка-Райта	19
1.4.2 Расширения алгоритма Кларка-Райта	20
1.4.3 Последовательный алгоритм вставки Моля-Джеймсона	22
1.4.4 Последовательный алгоритм вставки Кристофидеса- Мингоzzi-Тосса	23
1.5 Двухфазные классические алгоритмы	24
1.5.1 Алгоритм заметания	25
1.5.2 Алгоритм Фишера-Джекумера	26
1.5.3 Алгоритм Брамела-Симчи-Леви	26
1.5.4 Алгоритм лепестков	27
1.5.5 Методы с решением ЗК перед кластеризацией	30
1.6 Классические улучшающие алгоритмы	31
1.6.1 Оптимизация отдельного маршрута	31
1.6.2 Алгоритмы для улучшения нескольких маршрутов	32
1.7 Метаэвристики	33
1.8 Алгоритм Османа поиска с исключениями	34
1.8.1 Понятие окрестности решения	35
1.8.2 Стратегия запрещения	35
1.8.3 Стратегия освобождения удаления из списка исключений	36
1.8.4 Стратегия выбора	36
1.8.5 Специальная структура данных для стратегии “наилуч- ший подходящий”	37
1.8.6 Функция для определения длины списка исключений	38
1.8.7 Критерий останова	38
1.8.8 Общий вид алгоритма	39
1.9 Другие варианты поиска с исключениями	39
1.9.1 Алгоритм Генро-Герца-Лапорте	40

1.9.2	Алгоритм Тейлорда	41
1.9.3	Алгоритм Ксю-Келли	42
1.9.4	Алгоритм Ригго-Рокарола	42
1.10	Моделируемый отжиг	43
1.10.1	Ранние алгоритмы моделируемого отжига для решения ЗМТ	44
1.10.2	Алгоритм Османа	44
1.11	Детерминированный отжиг	46
1.12	Генетический алгоритм	46
1.12.1	Основной вид генетического алгоритма	47
1.12.2	Применение генетического алгоритма для задач упоря- дочивания	48
1.12.3	Применение алгоритма для решения ЗМТ	48
1.13	Алгоритм на основе муравьиных колоний	50
1.14	Нейронные сети	52
1.15	Выводы	55
2	Сбалансированные алгоритмы решения ЗМТ	57
2.1	Сбалансированная ЗМТ	59
2.2	Вычисление количества вершин для каждого транспортного средства в СЗМТ	60
2.3	Общий вид процедуры дихотомического деления вершин на группы	62
2.4	Процедура дихотомической кластеризации для СЗМТ	63
2.5	Другие варианты дихотомического алгоритма для СЗМТ	67
2.6	Обменная оптимизация при дихотомическом делении	67
2.7	Алгоритм разрезания общего маршрута для СЗМТ	68
2.8	Алгоритм заметания для СЗМТ	70
2.9	Вычислительный эксперимент для СЗМТ	72
2.10	Сбалансированный алгоритм кластеризации для ЗМТУГ	77
2.11	Рекурсивная процедура дихотомического сбалансированного разделения вершин	79
2.12	Вычислительный эксперимент для ЗМТУГ	84
2.13	Дополнительные варианты сбалансированного алгоритма для ЗМТУГ	88

2.13.1	Вариант сбалансированного алгоритма с бэктрекингом	91
2.13.2	Ограничение количество вершин в маршрутах	93
2.14	Вариант сбалансированного алгоритма для нескольких депо	95
2.15	Использование геометрической информации для процедуры деления вершин	97
2.16	Выводы	101
3	Реализация алгоритмов решения ЗМТ	102
3.1	Основные понятия	102
3.2	Обработка несимметричных матриц	105
3.3	Пользовательский интерфейс	106
3.4	Интерфейс библиотеки для решения ЗМТ	108
3.4.1	Описание функции SolveBCVRP	113
3.4.2	Описание функции SolveVRP	115
3.5	Внутренняя структура библиотеки алгоритмов решения ЗМТ	116
3.6	Выводы	121
	Заключение	124
	Список литературы	126
	Приложение	136

Введение

Актуальность работы. Одним из способов экономии ресурсов при транспортировке грузов является применение систем поддержки принятия решений в области транспортной логистики. Разработка программных пакетов, решающих задачи этой отрасли, требует проведения серьёзных научных исследований с целью получения эффективных алгоритмов, пригодных для применения в повседневной практике.

Одной из ключевых функций систем поддержки принятия решений в области транспортной логистики является возможность расчёта и построения эффективных с точки зрения стоимости объезда маршрутов различного назначения на транспортной сети. Работа посвящена исследованию одной из таких задач, состоящей в нахождении маршрутов для посещения заданного множества адресов некоторым количеством единиц транспортных средств с обязательным возвращением в начальное местоположение после окончания поездки. Такая проблема наиболее часто встречается среди компаний, выполняющих развозку товара с некоторого склада до точек потребления или розничной торговли.

Математическая формулировка этой задачи широко известна как задача маршрутизации транспорта (ЗМТ). Существует ряд разновидностей ЗМТ с различными дополнительными условиями, позволяющими учитывать грузоподъёмность транспортных средств и другие ограничения для более полного представления деталей реальной действительности. ЗМТ является обобщением известной задачи коммивояжёра (ЗК) на случай построения сразу нескольких замкнутых маршрутов, проходящих через некоторую общую вершину, называемую депо. ЗМТ и ЗК принадлежат к классу задач дискретной оптимизации и являются NP-трудными. Не существует методов нахождения их точных решений и проверки оптимальности приближённых за полиномиальное время. Известен точный алгоритм решения ЗМТ на основе метода ветвей и границ, но в силу чрезмерно быстрого роста времени вычислений его невозможно применять для задач с более чем 25–30 вершинами.

Один из первых приближённых алгоритмов решения ЗМТ был предложен в 1964 г. (G. Clarke и J. W. Wright). В 1970-х и 1980-х годах исследования были продолжены, и полученные результаты составили группу так называемых классических алгоритмов (J. .B. Bramel, N. Christofides, B. .E. Gillett,

Ж. Renaud и др.). Эти алгоритмы заложили основные типы подходов к приближенному решению ЗМТ. С середины 1990-х годов исследования сосредоточились в направлении так называемых метаэвристик. Название метаэвристик указывает на то, что они не являются законченными эвристиками, готовыми для применения, а только представляют собой некоторый метод для построения законченной эвристики для конкретной задачи. Большинство из них основаны на наблюдениях за явлениями живой и неживой природы. Важной их особенностью является способность к преодолению точки локального минимума для продолжения поиска, поэтому потенциально они способны находить более качественные решения по сравнению с классическими эвристиками. Наибольший интерес вызывают следующие методы: поиск с исключениями (M. Gendreau, A. Hertz, G. Laporte, I. H. Osman и др.), моделируемый и детерминированный отжиг (I. H. Osman и др.), генетический алгоритм (J. L. Blanton, G. Jeon и др.), алгоритм на основе муравьиных колоний (B. Bullnheimer и др.) и нейронные сети (Y. Matsuyama и др.). В последние десять лет исследования уклонились в основном в сторону обработки сложных видов ограничений. В отечественной научной литературе решением задач маршрутизации транспорта занимались: А. О. Алексеев, М. Ю. Ахлебинский, И. И. Меламед, С. И. Сергеев, И. Х. Сигал и др.

Большое количество работ, посвящённых метаэвристикам, создали ситуацию неопределённости в научном мире, не позволяя однозначно определить наилучший алгоритм для практического внедрения. Метаэвристики содержат дискретные и непрерывные параметры, управляющие их работой и требующие выполнения процедуры вариации значений для получения удачной законченной эвристики. Подбор параметров необходимо выполнять не только для разных типов задач, но зачастую даже для каждого нового набора входных данных. Если поиск с исключениями содержит только 1–2 дискретных параметра, то моделируемый отжиг содержит ряд непрерывных параметров, делающих процедуру вариации практически невозможной. Генетический алгоритм и алгоритм на основе муравьиных колоний в процессе работы обрабатывают очень большой массив данных, приводящий к длительным вычислениям, а также содержат большое количество управляющих параметров.

В настоящее время не существует формализованного способа получения конкретных алгоритмов из метаэвристик, необходимого для автоматизации их применения в программных пакетах. Использование эмпирических фор-

мул не гарантирует получения наилучшего значения параметров, подходящего для обработки некоторого конкретного набора входных данных. Длительные вычисления в ходе работы метаэвристик также усложняют ситуацию. Поиск алгоритмов, дающих достаточно качественные решения, но в то же время свободных от влияния управляющих параметров и при этом быстрых, способных за разумное время находить маршруты для 1000000 и более вершин, является актуальной задачей. Необходимость решать задачи такого размера обусловлена масштабами транспортных магистралей современных крупных городов. Решение этой задачи позволит создавать программные приложения для массового применения, ориентированные на широкий круг пользователей.

Целью настоящей работы является получение приближённых алгоритмов решения ЗМТ, способных выполнять построение маршрутов для входных данных, содержащих до 1000 вершин и более при использовании матрицы стоимостей переездов и до 1000000 вершин при использовании геометрической информации. В рамках указанной цели были поставлены следующие задачи:

1. Разработка и исследование эффективных приближённых алгоритмов, дающих сбалансированное по количеству вершин в отдельных маршрутах решение ЗМТ при заранее указанном количестве транспортных средств.
2. Разработка и исследование эффективных приближённых алгоритмов, дающих решение ЗМТ с учётом грузоподъёмности транспортных средств и/или с ограничением количества вершин в одном маршруте, а также в случае нескольких депо и для случая несимметричной матрицы расстояний.

Методика исследования. В ходе исследования были использованы методы дискретной оптимизации, теории сложности алгоритмов, математического моделирования, математической статистики и объектно-ориентированного программирования.

Научная новизна работы.

1. Предложена модификация приближённого метода двухфазного решения ЗМТ, использующая на этапе кластеризации алгоритм сбалансированного дихотомического деления вершин на группы и позволяющая строить приближённые алгоритмы, обладающие трудоёмкостью порядка $O(n^2)$, показывающие при математическом моделировании снижение времени

работы и улучшения качества решений для задач, содержащих 200 вершин и более, по сравнению с распространёнными метаэвристиками. На основе предложенной модификации приближённого метода двухфазного решения ЗМТ разработаны и реализованы алгоритмы решения сбалансированной ЗМТ, ЗМТ с учётом грузоподъёмности и/или с ограничением количества вершин в каждом маршруте, ЗМТ для нескольких депо.

2. Предложена модификация алгоритма сбалансированного деления вершин на группы с использованием геометрической информации, обладающая трудоёмкостью $O(n \log^2 n)$ и позволяющая решать сбалансированную ЗМТ, ЗМТ с учётом грузоподъёмности и/или с ограничением количества вершин в каждом маршруте, ЗМТ с несколькими депо для входных данных, содержащих до 1000000 вершин.
3. Получена оценка погрешности замены несимметричной матрицы стоимостей переездов симметричной, необходимой для применения на практике известных и предложенных приближённых алгоритмов решения различных видов ЗМТ, не способных работать с несимметричной матрицей.

Теоретическая значимость работы заключается в следующем:

1. Предложенный в работе принцип сбалансированного дихотомического деления вершин на группы позволил разработать новый класс алгоритмов решения ЗМТ, обладающих низкой трудоёмкостью и высоким качеством решения в широком диапазоне объёмов входных данных.
2. Предложенный алгоритм способен послужить основой для разработки специализированных алгоритмов решения таких видов ЗМТ, как ЗМТ с ограничением на время доставки, ЗМТ с сопутствующим перевозчиком, ЗМТ с обратным грузом.

Практическая ценность диссертационной работы заключается в том, что созданное программное приложение пригодно для использования конечными пользователями для практического расчёта эффективных маршрутов в области транспортной логистики для задач, содержащих до 1000000 вершин и более.

Достоверность полученных результатов подтверждается анализом исследуемых алгоритмов и результатов вычислительного эксперимента с применением методов математической статистики.

Внедрение результатов работы. Результаты диссертационной работы в виде созданного программного приложения внедрены в промышленную геоинформационную систему IndorGIS.

Основные защищаемые положения:

1. Модификация приближённого метода двухфазного решения ЗМТ на основе принципа сбалансированного дихотомического деления вершин на группы, а также алгоритмы, созданные на её основе, для решения сбалансированной ЗМТ, ЗМТ с учётом грузоподъёмности и/или с ограничением количества вершин в каждом маршруте, ЗМТ для нескольких депо.
2. Модификация алгоритма сбалансированного деления вершин на группы с использованием геометрической информации для решения различных видов ЗМТ.
3. Оценка погрешности замены несимметричной матрицы стоимостей перевозок симметричной.
4. Реализация предложенных алгоритмов решения различных видов ЗМТ в виде библиотеки классов.

Апробация диссертационной работы. Основные положения и отдельные результаты диссертационной работы докладывались и обсуждались на следующих конференциях:

1. XLV Международной научно-студенческой конференции “Информационные технологии” в г. Новосибирске в 2007 г.
2. Международной научно-практической конференции “Информационные технологии и математическое моделирование” в г. Анжеро-Судженске в 2007 г.
3. VII всероссийской научно-практической конференции с международным участием “Информационные технологии и математическое моделирование” в г. Анжеро-Судженске в 2008 г.
4. VIII всероссийской научно-практической конференции с международным участием “Информационные технологии и математическое моделирование” в г. Анжеро-Судженске в 2009 г.

Публикации. По теме диссертации опубликовано шесть печатных работ, в том числе одна статья [11] в журнале из списка ВАК.

Личный вклад автора. Постановка задачи, планирование основных путей её решения и обсуждение результатов осуществлялись совместно с научным руководителем. Разработка и исследование алгоритмов, проведение вычислительного эксперимента, реализация и отладка программного обеспечения осуществлялись автором самостоятельно.

Гл. 1 диссертации содержит обзор различных постановок ЗМТ и наиболее известных приближённых алгоритмов её решения. Кроме общего вида задачи, приводятся описания ЗМТ с учётом грузоподъёмности транспортных средств, ЗМТ с ограничением количества вершин в каждом маршруте, а также развитие ЗМТ для случая нескольких депо. Упоминаются некоторые дополнительные виды задачи, получившие распространение на практике, но не рассматриваемые детально в работе. В разд. 1.3 приведена детальная классификация известных приближённых алгоритмов решения ЗМТ с общей характеристикой каждой группы. Точные методы решения этой задачи не упоминаются в силу неприемлемо больших затрат вычислительных ресурсов для их применения. Описаны следующие алгоритмы: алгоритм Кларка-Райта с некоторыми расширениями, последовательный алгоритм вставки Моля-Джеймсона, последовательный алгоритм вставки Кристофидеса-Мингоззи-Тосса, алгоритм заметания, алгоритм Фишера-Джекумера, алгоритм Брамела-Симчи-Леви, алгоритм лепестков, а также приведены некоторые идеи подходов, выполняющих последовательное улучшение уже найденного решения. Перечисленные алгоритмы составляют группу так называемых классических эвристик. Известен ряд новых идей, традиционно относимых к другой группе метаэвристик. Метаэвристики являются общими методами решения задач комбинаторной оптимизации, широко известными и за пределами области алгоритмов решения ЗМТ. Их применение для ЗМТ показывает неплохие результаты, но затруднено на практике из-за недостаточно конкретной формулировки. Рассматриваются поиск с исключениями, моделируемый и детерминированный отжиг, генетический алгоритм, алгоритм на основе муравьиных колоний и нейронные сети.

В гл. 2 изложено основное содержание проведённых исследований. Предлагается новая формулировка ЗМТ с наложенным условием сбалансирования, далее называемая сбалансированной ЗМТ. Она требует, чтобы количество вершин для каждой пары построенных маршрутов не отличалось бы более, чем на единицу. На основе нового вида задачи приведён алгоритм,

выполняющий разделение вершин на группы для каждого будущего маршрута при помощи сбалансированной дихотомической процедуры. Для получения окончательного решения ЗМТ требуется решить ЗК внутри каждой группы. Кратко перечислены направления поиска, показавшие во время исследований ухудшение результатов, а также рассмотрено влияние условия сбалансирования на такие известные методы, как алгоритм заметания и разрезание общего маршрута. В каждом случае показано уменьшение времени работы алгоритмов. Приводятся результаты проведённого вычислительного эксперимента для алгоритмов решения сбалансированной ЗМТ. Развитие идеи сбалансированного дихотомического деления позволило создать новый алгоритм решения стандартной ЗМТ с учётом грузоподъёмности, показывающий хорошие результаты качества для больших наборов входных данных при сравнительно непродолжительном времени работы. Проведён вычислительный эксперимент, сравнивающий новый алгоритм с алгоритмом Османа поиска с исключениями. В вычислительном эксперименте была рассмотрена возможность запуска алгоритма Османа поверх решения, полученного алгоритмом сбалансированного дихотомического деления. Новый алгоритм показывает хорошие результаты для наборов данных с более, чем 150 вершинами. После вычислительного эксперимента описаны две разновидности сбалансированного дихотомического алгоритма: вариант с бэктрекингом, показывающий уменьшение стоимости решений в ущерб равномерности загрузки транспортных средств, и вариант, позволяющий задать явное ограничение максимального количества вершин в каждом маршруте. Дальнейшее развитие идеи сбалансированного деления позволило создать алгоритм, решающий ЗМТ для нескольких депо. Приводятся результаты вычислительного эксперимента с запуском алгоритма для задач с двумя и четырьмя депо.

Гл. 3 посвящена описанию реализации исследованных алгоритмов в виде динамически подключаемой библиотеки с целью последующей интеграции в геоинформационную систему. Перечислены основные этапы решения ЗМТ на практике: загрузка данных карты из внешнего файла, построение транспортного слоя, построения графа для получения подходящей математической модели транспортной сети и вычисления матрицы стоимости переездов, выбор желаемых пунктов обслуживания и депо, запуск алгоритмов и оценивание результатов. Подробно описаны особенности каждого этапа и связанные с ним понятия. Приведена оценка перехода к использованию симметричной

матрицы стоимости переездов взамен несимметричной, вызванного неспособностью работы большей части алгоритмов с несимметричными матрицами. Детально описывается интерфейс пользователя приложения, позволяющий в удобной форме выполнять процедуру построения маршрутов для решения практических задач. Оставшаяся часть этой главы посвящена описанию интерфейса созданной библиотеки и структур данных, использованной при реализации алгоритмов.

В заключении приведены и описаны основные результаты работы. Приложение содержит акт о внедрении подготовленной реализации полученных алгоритмов.

1 Обзор алгоритмов решения ЗМТ

В этой главе произведём подробный обзор ЗМТ: сформулируем её точную постановку, приведём классификацию известных методов и опишем наиболее распространённые из них.

1.1 Терминология

Несмотря на то, что ЗМТ является развитием ЗК, и эти задачи родственны, необходимо заметить, что в ЗМТ принята терминология, несколько отличная от ЗК. Если в ЗК обычно употребляют такие понятия как “коммивояжёр” (travelling-salesman), “город” (town) и “путь” (tour), то в ЗМТ “коммивояжёр” заменён на “экипаж” (vehicle) или “транспортное средство”, “город” — на “вершину” (vertex), “путь” — на “маршрут” (route). В некоторых случаях термин “вершина” заменяется синонимом “клиент” (customer). Кроме этого, в ЗМТ появляется ещё один дополнительный термин — “депо” (depo).

На практике депо — это некоторая особая вершина, где начинаются и заканчиваются маршруты всех транспортных средств. Обычно депо представляет из себя склад для хранения продукции, предназначенной для развозки и доставки клиентам. Во всех вариантах ЗМТ предполагается наличие не менее одного депо. Отдельно выделяется разновидность задачи, в которой рассматриваются случаи с более чем одним депо.

Если в задаче только одно депо, то все экипажи должны начинать путь с него и заканчивать его в этом же депо. Другими словами, все маршруты должны включать в себя вершину депо. Если же депо несколько, то каждый маршрут должен содержать вершину только одного из них.

В задаче присутствует такое понятие, как стоимость объезда последовательности вершин транспортным средством. Это ключевой параметр, требующий минимизации. На практике он выражает любые затраты посещения клиентов и может представлять как стоимость потребляемого топлива, так и время, требуемое для выполнения работы. Для исследования алгоритмов не важно, что за ним скрывается в действительности, примем его как обобщение всех видов затрат на передвижение. Единственное ограничение связано с тем, что стоимость проезда не может быть отрицательной.

1.2 Постановка ЗМТ

Прежде всего сформулируем общую постановку ЗМТ. Она выглядит следующим образом [26]:

1. Задаётся $V = \{v_0, v_1, \dots, v_n\}$ — множество всех вершин. v_0 — вершина, в которой построенные маршруты должны начинаться и заканчиваться. Назовём её “депо”.
2. $V' = V \setminus \{v_0\}$ — множество из n целевых вершин для посещения.
3. Задаётся C — матрица стоимостей переезда между вершинами; c_{ij} — стоимость переезда между вершинами v_i и v_j .
4. Требуется построить m маршрутов транспортных средств минимальной суммарной стоимости, которые начинаются и заканчиваются в депо v_0 , и каждая вершина из V' должна быть включена в маршрут одного и только одного транспортного средства.

Известна задача, называемая задачей k -коммивояжёров [1, 5, 6], но в своей постановке совпадающая с постановкой ЗМТ. Количество маршрутов m в разных вариантах задачи может быть задано заранее, либо допускается его автоматическое определение в ходе вычислений. Вариант с определением в ходе вычислений обычно используется при наличии дополнительных ограничений (см. ниже). Если m задано заранее и равно единице, то получаем стандартный вид ЗК.

Матрица C может быть несимметричной, но должна удовлетворять условию треугольника, определяя метрическое пространство расстояний. В этом случае говорят о несимметричной ЗМТ. Если C симметрична, то задачу называют симметричной. В этом случае также необходимо выполнение условия треугольника.

Общий вид ЗМТ не всегда достаточно детально моделирует практические задачи, поэтому существует ряд дополнительных модификаций. Одна из самых важных — вариант ЗМТ с учётом грузоподъёмности транспортных средств (ЗМТУГ). Она имеет следующий вид:

1. Задаётся $V = \{v_0, v_1, \dots, v_n\}$ — множество всех вершин. v_0 — вершина, в которой построенные маршруты должны начинаться и заканчиваться. Назовём её “депо”.

2. $V' = V \setminus \{v_0\}$ — множество из n целевых вершин для посещения.
3. Задаётся C — матрица стоимостей переезда между вершинами (симметричная или не симметричная); c_{ij} — стоимость переезда между вершинами v_i и v_j .
4. Задаётся величина $q > 0$, определяющая грузоподъёмность каждого транспортного средства.
5. Задаётся множество чисел $Q = \{q_1, \dots, q_n\}$, где q_i определяет величину потребности в товаре вершины v_i . q_i должно быть строго больше нуля.
6. Требуется построить m маршрутов транспортных средств минимальной суммарной стоимости, которые начинаются и заканчиваются в депо v_0 , и каждая вершина из V' должна быть включена в маршрут одного и только одного транспортного средства. Количество маршрутов m определяется в ходе вычислений. Сумма потребности в товаре всех вершин-клиентов каждого маршрута не должна превышать величину q .

Существует разновидность ЗМТ с ограничением количества вершин-клиентов (ЗМТОКВ) в каждом маршруте. Если это условие сочетается с наложенным ограничением грузоподъёмности транспортных средств, задача приобретает вид:

1. Задаётся $V = \{v_0, v_1, \dots, v_n\}$ — множество всех вершин. v_0 — вершина, в которой построенные маршруты должны начинаться и заканчиваться. Назовём её “депо”.
2. $V' = V \setminus \{v_0\}$ — множество из n целевых вершин для посещения.
3. Задаётся C — матрица стоимостей переезда между вершинами (симметричная или не симметричная); c_{ij} — стоимость переезда между вершинами v_i и v_j .
4. Задаётся величина $q > 0$, определяющая грузоподъёмность каждого транспортного средства.
5. Задаётся множество чисел $Q = \{q_1, \dots, q_n\}$, где q_i определяет величину потребности в товаре вершины v_i . q_i должно быть строго больше нуля.
6. Задаётся величина $d > 0$, определяющая максимальное количество вершин-клиентов в каждом маршруте.

7. Требуется построить m маршрутов транспортных средств минимальной суммарной стоимости, которые начинаются и заканчиваются в депо v_0 , и каждая вершина из V' должна быть включена в маршрут одного и только одного транспортного средства. Количество маршрутов m определяется в ходе вычислений. Сумма потребности в товаре всех вершин-клиентов каждого маршрута не должна превышать величину q . Количество вершин-клиентов в каждом маршруте не должно превосходить d .

Известна разновидность задачи, в котором единственное депо заменено на множество из нескольких возможных депо. Каждый маршрут может начинаться в любом из них, заканчиваться должен обязательно в том, где начался. Приведём общий вид такого варианта задачи без ограничений на грузоподъёмность или максимальное количество вершин-клиентов, хотя они также могут присутствовать:

1. Задаётся множество $V = \{v_1, \dots, v_n\}$ из n вершин-клиентов, и множество $D = \{d_1, \dots, d_k\}$ — множество из k вершин-депо.
2. Задаётся C — матрица стоимостей переезда (симметричная или несимметричная) между вершинами, включая вершины-клиенты и вершины-депо.
3. Требуется построить m маршрутов транспортных средств минимальной суммарной стоимости, которые начинаются и заканчиваются в одном из депо, заданных множеством D , и каждая вершина из V должна быть включена в маршрут одного и только одного транспортного средства. Количество маршрутов m может быть заданным заранее или вычисляться в ходе работы.

Помимо приведённых иногда выделяют следующие дополнительные разновидности задачи:

1. ЗМТ с разделённой доставкой: в этом ослаблении задачи допускается посещение вершин не одним, а несколькими транспортными средствами.
2. Стохастическая ЗМТ: некоторый параметр входных данных может иметь случайный вид, заданный с некоторой вероятностью.
3. ЗМТ с обратным грузом: транспортным средствам ставится задача не только доставлять груз к клиентам, но и забирать товар от них.

4. ЗМТ с сопутствующим перевозчиком: каждому транспортному средству даётся сопутствующий перевозчик, который следует вместе с ним до определённого момента для увеличения количества развозимого товара, а затем возвращается в депо.
5. ЗМТ с окнами времени: для каждой целевой вершины задаётся некоторый отрезок времени, в который она может принять транспортное средство.

1.3 Классификация алгоритмов для решения ЗМТ

К настоящему моменту известно достаточно много алгоритмов для решения ЗМТ. Большею частью это эвристические методы, используемые при наличии матрицы расстояний или информации о расположении вершин на плоскости. Как говорилось во введении, ЗМТ является NP-трудной задачей, поэтому наиболее интенсивно поиск ведётся в направлении приближённых алгоритмов. Предлагались точные методы решения ЗМТ, как, например, метод ветвей и границ, но время вычислений при их применении растёт слишком быстро. ЗМТ предполагает значительно большее количество вариантов решений для просмотра, чем ЗК при одинаковом количестве вершин, в то время как применение метода ветвей и границ для решения ЗК уже затруднительно для наборов данных с 30 вершинами и больше.

Известные подходы обычно ориентируются на общую формулировку ЗМТ, в которой предполагается симметричная или несимметричная матрица расстояний, не заданное жёстко количество транспортных средств, и отслеживается только ограничение по их грузоподъёмности или максимальной длине маршрута. В описаниях известных алгоритмов, приводимых ниже, указываются дополнительные уточнения постановки задачи в тех местах, где это важно.

Поиск решений ЗМТ начался в 60-е годы XX века. Эвристические методы, которые в наши дни называют классическими, разработаны в основном между 1960 и 1990 годом. В последние двадцать лет усилия направлены в основном на направление так называемых метаэвристик.

Особенность метаэвристических алгоритмов в том, что они не дают точного описания порядка действий для решения задачи, и каждый из них должен быть дополнительно конкретизирован путём подбора значений управляющих

параметров. Авторы метаэвристических алгоритмов приводят константы, дающие по их мнению наиболее удачные результаты, но в некоторых ситуациях оказывается возможным найти более качественные решения, если провести дополнительные исследования влияния параметров.

Классические алгоритмы можно разбить на три группы [66]:

1. **Конструктивные алгоритмы:** выполняют постепенное построение решения, отслеживая рост его стоимости, но не имеют фазы дальнейшего улучшения.
2. **Двухфазные (кластерные) алгоритмы:** задача разбивается на две части — группировку вершин для каждого будущего маршрута (кластеризацию) и решение ЗК для каждой полученной группы. Возможно наличие обратной связи между этапами решения. В свою очередь двухфазные методы делятся ещё на две группы:
 - (а) сначала кластеризация, затем поиск решения ЗК;
 - (б) сначала решение ЗК, а затем разделение на несколько маршрутов. ЗК решается для всех вершин исходного множества.
3. **Улучшающие алгоритмы:** сначала ведётся поиск некоторого решения, а затем делаются попытки обмена вершин (рёбер) внутри каждого маршрута или между маршрутами.

Следующие алгоритмы относят к разряду метаэвристик [43]:

1. Поиск с исключениями.
2. Моделируемый отжиг.
3. Детерминированный отжиг.
4. Генетический алгоритм.
5. Алгоритм на основе муравьиных колоний.
6. Нейронные сети.

Все метаэвристики, кроме поиска с исключениями, предложены на основе идей, появившихся при наблюдении процессов живой и неживой природы. Первые три подхода: поиск с исключениями, моделируемый отжиг и детерминированный отжиг начинают работу с некоторого начального решения x_1 ,

на каждой итерации t выполняют переход от решения x_t к решению x_{t+1} , находящимся в окрестности $N(x_t)$ решения x_t , до тех пор, пока не будет выполнено некоторое условие останова вычислений.

Если $f(x)$ обозначает стоимость решения x , то $f(x_{t+1})$ в метаэвристиках не обязательно меньше $f(x_t)$ [43]. Это является ключевым отличием метаэвристик от классических эвристик, применяемых для решения ЗМТ. В каждом случае такая возможность требует наличия дополнительного механизма защиты от возможного заикливания процесса поиска.

Генетический алгоритм проверяет на каждом шаге популяцию решений. В нём каждая новая популяция наследуется от предыдущей путём комбинирования её наилучших решений и удаления неудачных. Поиск с исключениями, генетический алгоритм и алгоритм на основе муравьиных колоний накапливают информацию по мере работы и используют её в дальнейших вычислениях. Нейронные сети — это самообучающийся метод, в котором ведётся подгонка набора весовых коэффициентов до тех пор, пока не будет найдено подходящее решение.

1.4 Конструктивные классические алгоритмы

1.4.1 Алгоритм Кларка-Райта

Алгоритм Кларка-Райта (Clarke and Wright) [27] — это один из самых известных алгоритмов для решения ЗМТ. Его идея основана на процессе слияния мелких маршрутов в более крупные, проводимого до тех пор, пока есть возможность уменьшить суммарную стоимость объезда. Особую роль в этом алгоритме играет понятие “сбережения” (saving) — это снижение общей стоимости решения, получаемое при объединении двух маршрутов. Рассмотрим ситуацию, когда маршрут $(0, \dots, i, 0)$ и маршрут $(0, j, \dots, 0)$ могут быть совмещены в единую последовательность $(0, \dots, i, j, \dots, 0)$. Сбережением является изменение расстояния, равное $s_{ij} = c_{i0} + c_{0j} - c_{ij}$, если оно больше нуля, где c_{ij} — расстояние между соответствующими вершинами. Алгоритм применяется в случаях, когда количество экипажей не определено заранее и его можно вычислять в ходе работы. Его можно использовать как для симметричных задач, так и для несимметричных, но в [101] показано, что качество работы для симметричных случаев заметно ухудшается. Известны два варианта реализации: параллельный и последовательный. В обоих случаях

для них требуется предварительное выполнение подготовительного этапа. Он заключается в:

1. В вычислении сбережений $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ для $i, j = 1, \dots, n$ и $i \neq j$.
2. Создании n маршрутов транспортных средств $(0, i, 0)$ для $i = 1, \dots, n$.
3. Сортировке сбережений в порядке убывания.

Рассмотрим сначала параллельный вариант алгоритма:

1. Просматриваем построенный список сбережений с его начала и выполняем следующие действия:
2. Для текущего элемента списка s_{ij} определяем, существуют ли два маршрута, один из которых содержит дугу или ребро $(0, j)$, второй — дугу или ребро $(i, 0)$, и которые могут быть соединены в один общий маршрут.
3. Если такие маршруты найдены, то выполняем их объединение, удаляя $(0, j)$ и $(i, 0)$ а затем добавляя (i, j) .

Последовательный вариант можно представить следующим образом:

1. Для всех маршрутов $(0, i, \dots, j, 0)$ выполним следующие действия:
2. Проведём поиск элемента в списке сбережений s_{ki} или s_{jl} , который может быть использован для объединения текущего маршрута с некоторым, содержащим дугу (ребро) $(k, 0)$ или $(0, l)$.
3. Если элемент был найден, то выполним объединение и применим процедуру ещё раз к новому маршруту.
4. Если сбережение найти не удалось, то перейдём к следующему маршруту и продолжим работу.
5. Завершим работу, когда объединения более невозможны.

Вычислительные результаты сравнения показывают, что параллельный вариант алгоритма даёт результаты лучше, чем последовательный.

1.4.2 Расширения алгоритма Кларка-Райта

Один из недостатков алгоритма Кларка-Райта заключается в том, что эффективность его работы падает по мере приближения к концу вычислений,

в то время как в начале работы решения получаются относительно удачные. В [42] и в [107] предлагалось обобщение понятия сбережения с целью устранения этого недостатка. Сбережение представляется в виде $s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij}$, где λ — параметр для учёта формы маршрута, который позволяет сделать акцент на расстояния между вершинами для соединения. В [56] показано, что $\lambda = 0,4$ улучшает решения, как с точки зрения суммарного расстояния, так и с точки зрения конечного количества маршрутов.

Алгоритм Кларка-Райта может оказаться неэффективным с точки зрения времени исполнения, т. к. во всех вариантах все выигрыши должны быть вычислены, сохранены и отсортированы. Различные авторы предпринимали попытки предложить расширения, уменьшающие время исполнения и использование памяти, необходимые для работы этого алгоритма. Большинство из них выполнено в 70-е годы, а также в начале 80-х годов, когда многие исследователи ещё работали с вычислительной техникой, значительно менее производительной, чем в наши дни.

При реализации стратегии, основанной на понятии сбережения, необходимо уделять внимание двум важным аспектам: методу нахождения элемента наибольшего сбережения в списке и требованиям к хранению данных в памяти. Здесь поиск наибольшего сбережения — это самый трудоёмкий процесс. Для решения этой задачи применяются три подхода:

1. Использование полного алгоритма сортировки. Например, сортировки Хоара.
2. Использование ограниченной сортировки при помощи построения кучи [56]. Здесь кучей называется специальное двоичное дерево, в котором сбережения расположены так, что родительский узел всегда имеет большее значение, чем дочерние. Когда происходит слияние маршрутов, то куча позволяет быстро и эффективно исключать соответствующие узлы.
3. Использование специального итеративного алгоритма для вычисления максимального сбережения [76]. В работе было показано, что $s_{ij} > \bar{s}$ всегда, когда $c_{0i} > 0,5\bar{s}$ и $c_{0j} > 0,5\bar{s}$, на основе того, что все расстояния положительные и выполняется правило треугольника в правой части определения сбережения, где \bar{s} — текущее максимальное значение сбережения. Приведённое необходимое условие затем используется для эффективного нахождения максимального сбережения.

В [31] и в [16] описана ещё одна интересная модификация алгоритма, основанного на понятии сбережения. На каждом шаге сбережение s_{pq} , полученное при слиянии маршрутов p и q , вычисляется как $s_{pq} = t(S_p) + t(S_q) - t(S_p \cup S_q)$, где S_k — множество вершин маршрута k , а $t(S_k)$ — длина оптимального решения ЗК, полученного на множестве S_k . Значение $t(S_k)$ может определяться разными способами. Один из них предложен на основе использования приближённой оценке длины точного решения.

Алгоритм Кларка-Райта утратил своё значение как самостоятельного подхода, т. к. предложено много более эффективных методов, но часто используется как способ нахождения начального решения для последующего улучшения. Например, он используется в алгоритме поиска с исключениями Османа, описанного ниже в разд. 1.8, который использовался для сравнительного тестирования сбалансированного алгоритма из разд. 2.10.

1.4.3 Последовательный алгоритм вставки Моля-Джеймсона

Алгоритм Моля-Джеймсона (Mole and Jameson) может применяться для задач с неопределённым заранее количеством транспортных средств. Он использует при расширении маршрута два параметра — λ и μ .

$$\alpha(i, k, j) = c_{ik} + c_{kj} - \lambda c_{ij},$$

$$\beta(i, k, j) = \mu c_{0k} - \alpha(i, k, j).$$

Работа алгоритма может быть представлена следующим образом:

1. Если нет неиспользованных вершин, завершаем работу алгоритма. В противном случае создаём новый маршрут $(0, k, 0)$, где k — любая не использованная вершина.
2. Вычисляем для любой неиспользованной вершины k стоимость возможной вставки в новый маршрут $\alpha^*(i_k, k, j_k) = \min\{\alpha(r, k, s)\}$, где r и s — любые две соседние вершины, принадлежащие последнему созданному маршруту. Вершины i_k и j_k — две вершины, соответствующие α^* . Если возможных вариантов вставки не существует, возвращаемся на шаг 1. В противном случае вершину для вставки k^* выберем такую, чтобы она давала $\beta^*(i_{k^*}, k^*, j_{k^*}) = \max\{\beta(i_k, k, j_k)\}$ среди всех возможных k . Добавим k^* между вершинами i_{k^*} и j_{k^*} .

3. После добавления вершины выполним оптимизацию при помощи процедуры 3-опт [67] и вернёмся на шаг 2.

Правила вставки контролируются параметрами λ и μ . Например, если $\lambda = 1$ и $\mu = 0$, алгоритм вставит вершину, позволяющую получить минимальное расстояние. Если $\lambda = \mu = 0$, то вставляемая вершина соответствует минимальной сумме расстояний между двумя соседними вершинами. Если $\mu = \infty$ и $\lambda > 0$, то вставляется вершина, максимально удалённая от депо.

1.4.4 Последовательный алгоритм вставки Кристофидеса-Мингоzzi-Тосса

Кристофидес, Мингоzzi и Тосс (Christofides, Mingozzi and Toth) разработали более глубокий последовательный алгоритм вставки [26]. Он также применяется для случаев с неопределённым заранее числом транспортных средств. Это двухфазовый эвристический метод, которым также можно управлять при помощи двух параметров λ и μ .

Первая фаза работы:

1. Зададим номер первого маршрута $k = 1$.
2. Выберем любую неиспользованную вершину i_k для начальной инициализации маршрута k . Для каждой неиспользованной вершины i вычислим $\delta_i = c_{0i} + \lambda c_{ii_k}$.
3. Пусть $\delta_{i^*} = \min_{i \in S_k} \{\delta_i\}$, где S_k — множество всех неиспользованных вершин, которые могли бы быть добавлены к маршруту k . Добавим вершину i^* к маршруту k и выполним оптимизацию при помощи процедуры 3-опт [67]. Будем повторять этот шаг до тех пор, пока существуют вершины, которые могли бы быть добавлены в маршрут k .
4. Если все вершины были использованы в маршрутах, то завершим работу алгоритма. В противном случае установим $k = k + 1$ и вернёмся на шаг 2.

Вторая фаза работы:

1. Создадим k маршрутов $R_t = (0, i_t, 0)$ ($t = 1, \dots, k$, где k — количество маршрутов, полученное в конце первой фазы работы. Пусть $J = \{R_1, \dots, R_k\}$.

2. Для каждого маршрута $R_t \in J$ и для каждой вершины i , ещё не ассоциированной с маршрутом, вычислим $\varepsilon_{ti} = c_{0i} + \mu c_{ii_t}$ и $\varepsilon_{t^*i} = \min_{t^*} \{\varepsilon_{ti}\}$. Припишем вершину i маршруту R_{t^*} и будем повторять этот шаг до тех пор, пока все вершины не будут принадлежать маршруту.
3. Возьмём любой маршрут $R_t \in J$ и установим $J := J \setminus \{R_t\}$. Для каждой вершины i , ассоциированной с маршрутом R_t , вычислим $\varepsilon_{t'i} = \min_{R_t \in J} \{\varepsilon_{ti}\}$ и $\tau_i = \varepsilon_{t'i} - \varepsilon_{ti}$.
4. Добавим в маршрут R_t вершину i^* , соблюдая $\tau_{i^*} = \max_{i \in S_t} \{\tau_i\}$, где S_t — множество неиспользованных вершин, ассоциированных с маршрутом R_t , которые могли бы быть добавлены в маршрут R_t . Выполним оптимизацию маршрута R_t при помощи процедуры 3-опт. Будем повторять этот шаг до тех пор, пока есть вершины, которые можно добавить в маршрут R_t .
5. Если $|J| \neq \emptyset$, то возвращаемся к шагу 2 второго этапа. В противном случае, если неиспользованных вершин больше нет, то завершаем работу. Если есть неиспользованные вершины, то создаём новые маршруты, начиная с шага 1 первого этапа.

1.5 Двухфазные классические алгоритмы

Двухфазные алгоритмы можно условно разделить на следующие группы:

1. Алгоритмы, которые сначала выполняют кластеризацию, т. е. разделяют множество вершин на группы для каждого транспортного средства, затем выполняют решение ЗК для каждой группы:
 - (а) самые простые, известные как “элементарные кластерные алгоритмы”, выполняющие одиночную операцию кластеризации, за которой следует решение ЗК внутри каждого кластера;
 - (б) Алгоритм лепестков (petal algorithm), который выполняет построение большого количества перекрывающихся кластеров (и маршрутов, с ними ассоциированных), а затем выбирает из них наилучшее подмножество.
2. Алгоритм, выполняющий разрезание общего маршрута, полученного путём решения ЗК для всех вершин на фрагменты, предназначенные для каждого транспортного средства.

К элементарным кластерным алгоритмам относятся алгоритм заметания, алгоритм Фишера-Джекумера и алгоритм Брамела-Симчи-Леви.

1.5.1 Алгоритм заметания

Алгоритм заметания (sweep algorithm) [49] используется для первоначальной обработки задач ЗМТ. В процессе его работы наполнение кластеров определяется поворотом луча, исходящего из депо. Затем для каждого кластера отдельно решается ЗК. В некоторых вариантах алгоритма присутствует фаза последующей оптимизации, в которой производится обмен вершинами между соседними кластерами, после чего выполняется корректировка маршрутов. Алгоритм не использует заранее заданного количества транспортных средств. Обратим внимание, что он использует информацию о расположении вершин на плоскости, что является выходом за пределы постановки задачи, приведённой в разд. 1.2. Первое упоминание этого алгоритма встречается в [104] и [105], но его обычно приписывают [49].

Пусть каждая вершина i представлена её полярными координатами (θ_i, ρ_i) , где θ_i — угол, ρ_i — длина радиуса. Зададим значение $\theta_{i^*} = 0$ произвольной вершине i^* и вычислим все оставшиеся углы относительно $(0, i^*)$. Расположим вершины по возрастанию θ_i (рис. 1).

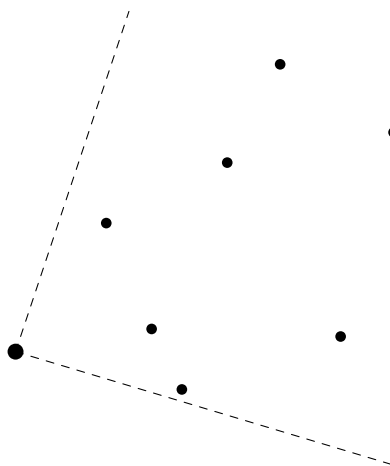


Рис. 1. Расположение вершин в секторе на плоскости относительно депо

Алгоритм выполняет следующие действия:

1. Возьмём неиспользованное транспортное средство k .
2. Начнём построение маршрута с вершины, имеющей наименьший угол и продолжим приписывание вершин транспортному средству k до тех пор,

пока не будет достигнута максимальная грузоподъёмность экипажа или любое другое ограничение.

3. Если неиспользованные вершины остались, то процедуру необходимо повторить для другого транспортного средства.
4. Для каждого транспортного средства выполним решение ЗК одним из точных или приближённых алгоритмов.

1.5.2 Алгоритм Фишера-Джекумера

Алгоритм Фишера-Джекумера (Fisher and Jaikumar) [39] использует для формирования кластеров не геометрический метод, а обобщённую задачу о назначениях. Количество транспортных средств k в этом методе предполагается заданным явно. Опишем работу алгоритма:

1. Выберем по одной вершине j_k из множества всех вершин V для начального заполнения каждого кластера k .
2. Вычисляем стоимость d_{ik} назначения каждой вершины i каждому кластеру k как $d_{ij_k} = \min\{c_{0i} + c_{ij_k} + c_{j_k0}, c_{0j_k} + c_{j_ki} + c_{i0}\} - (c_{0j_k} + c_{j_k0})$.
3. Решаем обобщённую задачу о назначениях со стоимостями d_{ij} , с учётом весов клиентов q_i и грузоподъёмности транспортных средств Q .
4. Решаем ЗК для каждого кластера на основе результатов, найденного при помощи обобщённой задачи о назначениях.

1.5.3 Алгоритм Брамела-Симчи-Леви

Алгоритм Брамела-Симчи-Леви (Bramel and Simchi-Levi) [23] — ещё один классический двухфазовый алгоритм. В её описании используется понятие “ядра кластера” (cluster seed), под которым подразумевается некоторая вершина, используемая для начального наполнения кластера.

В алгоритме Брамела-Симчи-Леви ядра кластеров определяются путём решения задачи размещения с ограничениями на мощности (capacitated location problem). Авторы предлагают определить K ядер кластеров из n местоположений клиентов. Последующее распределение оставшихся клиентов предлагается выполнять так, чтобы суммарное расстояние от всех клиентов до ближайших ядер кластеров было бы минимальным, при этом отслеживая, чтобы

общая потребность в товаре для каждого ядра не превышала установленную константу Q .

Рассмотрим частичный маршрут k , описываемый вектором $(i_0, i_1, \dots, i_l, i_{l+1})$. Пусть $T_k = \{i_1, \dots, i_l\}$ и $t(T_k)$ — длина точного решения ЗК на множестве T_k . Тогда стоимость добавления d_{ik} неиспользованной вершины i в маршрут k равна $d_{ik} = t(T_k \cup \{i\}) - t(T_k)$. Точное вычисление d_{ik} может оказаться очень трудоёмким, поэтому предлагаются две аппроксимации \bar{d}_{ik} :

1. Прямая стоимость: $\bar{d}_{ik} = \min_{h=1, \dots, l} \{2c_{ii_h}\}$.
2. Стоимость ближайшей вставки: $\bar{d}_{ik} = \min_{h=0, \dots, l} \{c_{i_h i} + c_{i i_{h+1}} - c_{i_h i_{h+1}}\}$.

Авторы показали, что алгоритм с использованием первого варианта получается асимптотически оптимальным.

1.5.4 Алгоритм лепестков

Алгоритм лепестков — это развитие идей метода заметания. Приводимое ниже описание [85] позволяет получать достаточно удачные решения ЗМТ. В своей работе он первоначально строит избыточное множество маршрутов и выполняет выборку из него при помощи решения задачи о разбиении множества.

Основное понятие этого алгоритма — это понятие r -лепестка (r -petal). Под r -лепестком здесь подразумевается множество из r маршрутов, которые в своей совокупности покрывают все вершины, расположенные на плоскости внутри некоторого сектора с центром в депо (рис. 2).

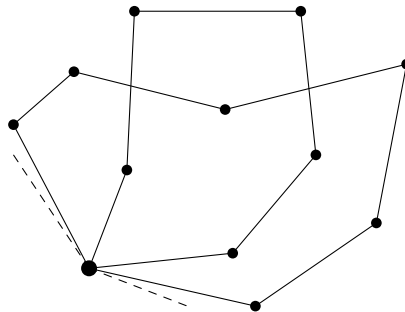


Рис. 2. Пример маршрутов 2-лепестка

На первом шаге работы необходимо сгенерировать множества r -лепестков, которые должны покрывать все вершины исходного набора данных. Ал-

горитм, описываемый здесь для примера, генерирует только 1-лепестки и 2-лепестки. Набор лепестков строится при помощи процедуры заметания (см. п. 1.5.1).

Процедура построения 1-лепестка на некотором множестве вершин S с включением в маршрут депо фактически является решением ЗК. Авторы алгоритма применяли для этих целей алгоритм I^3 [82]. I^3 — это трёхфазовый алгоритм приближённого решения ЗК, который первоначально строит некоторую оболочку для всех вершин (не обязательно выпуклую), а затем выполняет добавление вершин, не вошедших в полученное предварительное решение. Последней фазой работы является процедура оптимизации при помощи алгоритма 4-опт*. При экспериментальном тестировании этот алгоритм находил решения, в среднем уступающие оптимальным на 2,2%.

Процедура построения 2-лепестка на некотором множестве вершин S выглядит следующим образом: прежде всего выбираются две максимально удалённые вершины, которые используются для начального наполнения маршрутов, оставшиеся вершины затем добавляются по очереди в один из них по принципу самой выгодной вставки и с соблюдением наложенных ограничений. После нового добавления запускается процедура 4-опт* для оптимизации промежуточного решения. Возможно применение разных подходов для поиска двух начальных вершин и процедуры добавления. Например, метод нахождения двух самых удалённых точек может быть заменён поиском самого большого угла между вершинами множества S относительно депо, но при тестировании такие варианты не показывают заметных улучшений качества решений. Авторы исследовали применение различных методов вставки вершин в промежуточные маршруты. Вариант, в котором на каждом шаге вычислялась стоимость вставки всех вершин в отдельности с выбором наименьшей из них, приводил к получению сильно несбалансированных маршрутов. Особенно это было заметно на задачах с сильными ограничениями. В работе предлагаются исследования различных методов противодействия подобным эффектам, а также варианты более глубокой обработки ограничений.

Теперь рассмотрим процедуру, используемую для получения всех 1-лепестков и 2-лепестков. Без потери общности предположим, что все вершины пронумерованы в порядке увеличения угла в полярной системе координат с центром в депо. Договоримся, что $v_j = v_{j(\text{mod } n)}$, если $j > n$. Множество ле-

пестков строится при помощи следующих действий:

1. Установим $i = 0$.
2. Установим $i = i + 1$. Если $i > n$, то завершаем работу.
3. Определим множество вершин $S_{ii} = \{v_i\}$, запомним его и вычислим стоимость соответствующего начального маршрута $\bar{c}_{ii} = 2c_{0i}$. Установим $j = i + 1$. Определим множество вершин $S_{ij} = \{v_i, v_j\}$ и соответствующий маршрут (v_0, v_i, v_j, v_0) . Если он не удовлетворяет наложенным ограничениям, переходим на шаг 5. В противном случае запомним S_{ij} , соответствующий маршрут и его стоимость \bar{c}_{ij} .
4. Установим $j = j + 1$ и $S_{ij} = \{v_i, \dots, v_j\}$. Если общее количество товара для развозки по S_{ij} превышает предел грузоподъёмности, то переходим на шаг 5. В противном случае выполняем процедуру построения 1-лепестка при $S = S_{ij}$. Если не удаётся найти возможный 1-лепесток, то переходим на шаг 5. Если маршрут был найден, то запоминаем S_{ij} , соответствующее решение и его стоимость \bar{c}_{ij} . Повторим этот шаг с начала.
5. Если текущее количество товара для развозки по S_{ij} превосходит удвоенную максимальную грузоподъёмность, то переходим на шаг 6. В противном случае выполним процедуру построения 2-лепестка при $S = S_{ij}$. Если не удалось найти допустимый 2-лепесток, то переходим на шаг 6. Если 2-лепесток был найден, то запомним S_{ij} , соответствующее решение и его стоимость \bar{c}_{ij} . Установим $j = j + 1$ и $S_{ij} = \{v_i, \dots, v_j\}$. Повторим этот шаг сначала.
6. Некоторые из лепестков, только что созданных, могут доминировать. Если $j = 2$, переходим на шаг 2. В противном случае для $h = j, j - 1, \dots, 3$ рассмотрим множество вершин S_{ih} и последнюю вершину v_h , помещённую в S_{ih} . Пусть \bar{c}_{ih} — стоимость соответствующего лепестка. Если $\bar{c}_{i,h-1} \geq \bar{c}_{ih}$, то удаление v_h из S_{ih} должно приводить к лепестку, стоимость которого не превышает $\bar{c}_{i,h-1}$. Лепесток, определённый в $S_{i,h-1}$, затем замещается лепестком, определённым над $S_{ih} \setminus \{v_h\}$. Переходим на шаг 2.

Преимущество описанной процедуры перед алгоритмом, представленным в [40], заключается в том, что порождаемые ей пересекающиеся маршруты часто входят в оптимальное решение.

Осталось рассмотреть процедуру выбора подмножества лепестков для окончательного решения задачи. После того, как все 1-лепестки и 2-лепестки определены, оптимальная их комбинация может быть найдена путём решения задачи о разбиении множества.

Минимизируем

$$\sum_{l \in L} \bar{c}_l x_l,$$

соблюдая ограничения

$$\sum_{l \in L} a_{kl} x_l = 1 \quad (k = 1, \dots, n)$$

$$x_l = 0 \text{ или } 1 (l \in L),$$

где L — множество кандидатов r -лепестков ($r = 1$ или $r = 2$), \bar{c}_l — стоимость лепестка l , и $a_{kl} = 1$ тогда и только тогда, когда v_k принадлежит лепестку l .

Поставленная задача о разбиении множества может быть решена за полиномиальное время [87, 19, 99, 22]. В качестве дополнительной оптимизации можно провести проверку, нет ли возможности объединить некоторые маршруты, не нарушая наложенные ограничения. Если слияние некоторых маршрутов было проведено, то повторный запуск процедуры 1-лепестка позволит перестроить решение на множестве вершин из маршрутов, вошедших в слияние.

1.5.5 Методы с решением ЗК перед кластеризацией

Методы, в которых решение ЗК предшествует кластеризации, сначала решают ЗК для всего множества вершин без учёта ограничений, а затем выполняют декомпозицию полученного маршрута, на маршруты для всех транспортных средств. Этот подход применяется в случаях, когда количество экипажей не задано явно. Он впервые был предложен в [19], где показано, что второй этап решения является задачей о нахождении кратчайших путей в нециклическом графе и может быть решён за время $O(n^2)$. В алгоритме поиска кратчайших путей стоимость d_{ij} переезда между вершинами i и j равняется $c_{0i} + c_{0j} + l_{ij}$, где l_{ij} — стоимость переезда из вершины i в вершину j по решению ЗК. В [57] показано, что если клиенты имеют единичную потребность

в товаре, то этот алгоритм асимптотически оптимален. Тем не менее, это не справедливо для случаев произвольных значений потребности в товаре, за исключением некоторых тривиальных случаев [20].

1.6 Классические улучшающие алгоритмы

Улучшающие алгоритмы для ЗМТ обрабатывают либо отдельный маршрут за раз, либо несколько маршрутов. В случае для одного маршрута можно применять любые алгоритмы оптимизации для ЗК. Для второго варианта могут быть разработаны алгоритмы, которые анализируют структуру, представленную несколькими маршрутами.

1.6.1 Оптимизация отдельного маршрута

Большинство процедур оптимизации для ЗК могут быть описаны в терминах λ -опт операций, которые были предложены Лином [67]. В них λ рёбер удаляются из маршрута, и λ оставшихся сегментов пересоединяются во всех возможных комбинациях. Если улучшающее соединение найдено (первое достигнутое или наиболее удачное), то изменения вносятся в маршрут. Работа останавливается на локальном минимуме, если более невозможно найти подходящие варианты замен. Проверка λ -оптимальности решения требует времени $O(n^\lambda)$.

Предложены несколько вариантов этого подхода. Лин и Керниган [68] представили алгоритм, в котором λ изменяется динамически в процессе поиска. В [73] описан метод *Or*-опт, заключающийся в подмене последовательностей из 3, 2 или 1 соседних вершин на последовательность из другого фрагмента этого же маршрута. Фактически, он представляет собой ограниченную форму 3-опт оптимизации. Проверка *Or*-оптимальности требует времени $O(n^2)$. На той же основе в [82] предлагается ограниченная форма 4-опт алгоритма, называемая 4-opt* и требующая для проверки оптимальности времени $O(\omega n^2)$.

В [62] проведён анализ упомянутых методов, где авторы пришли к заключению, что оптимизация Лина-Кернигана даёт в среднем самые хорошие результаты.

1.6.2 Алгоритмы для улучшения нескольких маршрутов

В [98, 100, 64] приводятся алгоритмы, основанные на обмене рёбер между маршрутами. Они вобрали в себя различные схемы, предложенные ранее несколькими авторами [91, 34, 92, 38, 78, 74, 93].

Особое внимание уделяется общей схеме b -циклического k -переноса, в которой рассматривается циклическая перестановка b маршрутов, и выполняется перенос k вершин из каждого маршрута в следующий маршрут. Авторы показывают, что применение некоторых обменов b -циклического k -переноса (с $b = 2$ или с переменным значением b и $k = 1$ или $k = 2$) даёт интересные результаты.

В [98] перечислены основные варианты модификации маршрутов, Они являются частными случаями 2-циклических переносов.

1. Перекрещивание двух рёбер из двух маршрутов (рис. 3).
2. Обмен вершинами между двумя маршрутами (рис. 4).
3. Перенос вершин из маршрута в маршрут (рис. 5).
4. Комбинации предыдущих вариантов.

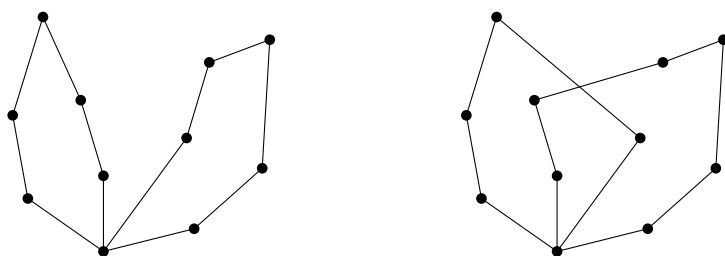


Рис. 3. Пересечение двух рёбер двух маршрутов

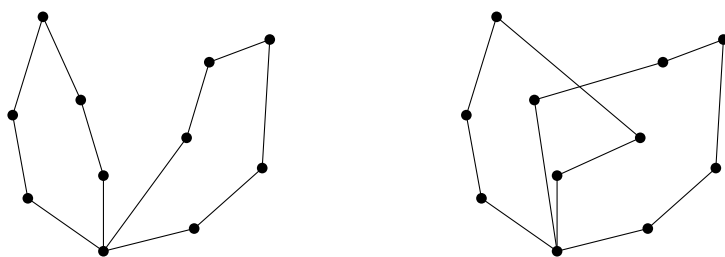


Рис. 4. Обмен вершинами двух маршрутов

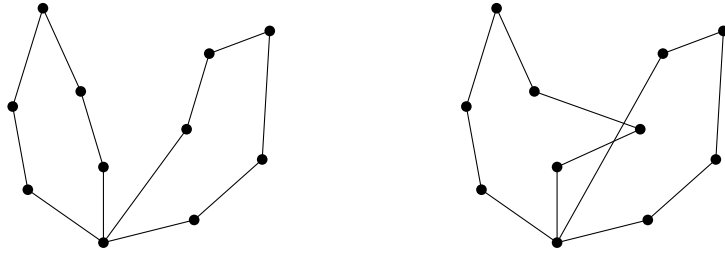


Рис. 5. Перенос вершин из маршрута в маршрут

1.7 Метаэвристики

Ниже рассматриваются наиболее интересные примеры так называемых метаэвристических алгоритмов. Все они, кроме поиска с исключениями, появились в ходе исследования различных явлений живой и неживой природы. Организация муравьиных колоний, наблюдения за размножением особей в некоторой популяции и изучение процессов при отжиге металлов послужили основой целой серии идей, применяемых для решения ЗМТ. Отличительной особенностью и в некоторой мере слабой стороной метаэвристических алгоритмов является наличие в их описании большого количества параметров и необходимость подбора их значений при практическом применении. В некоторых случаях процедуру подбора значений параметров необходимо выполнять непосредственно для каждого нового набора данных. Это заметно усложняет использование подобных алгоритмов для решения реальных задач на практике, т. к. требует очень высокой квалификации пользователя программных пакетов на их основе.

Сильной стороной рассматриваемых методов является возможность преодоления локального минимума в процессе поиска. При удачных обстоятельствах это позволяет просмотреть несколько точек локальных минимумов и выбрать из них наилучшую. Такая способность алгоритмов улучшает качество получаемых решений, хотя это приводит к росту требуемого процессорного времени.

Описываемые метаэвристические алгоритмы вызывают интерес с разных точек зрения. Поиск с исключениями является одним из самых исследованных подходов, но это не делает остальные идеи менее привлекательными. Согласно мнению исследователей, о чём ещё пойдёт речь ниже, такие методы, как генетические алгоритмы и алгоритмы на основе муравьиных колоний, в настоящий момент ещё относительно слабо изучены, хотя содержат боль-

шой простор для дальнейшего поиска улучшений, потенциально способных привести к заметному росту качества. Генетический алгоритм, в частности, использует в своей работе очень мало информации о структуре задачи, и это позволяет строить на его основе методы для решения ЗМТ с очень сложным набором ограничений, с которым не смогут справиться алгоритмы, дающие сейчас наилучшие результаты.

1.8 Алгоритм Османа поиска с исключениями

Поиск с исключениями (Taboo search) — это стратегия для решения задач комбинаторной оптимизации. Она основана на общих идеях, описанных в [50, 51, 52, 53]. Осман (Osman) удачно её применил [74] для решения ЗМТ.

Поиск с исключениями, как и моделируемый отжиг (см. разд. 1.10), способен преодолевать точку локального минимума. Метод, позволяющий это делать, основан на использовании так называемой оценочной функции, дающей характеристику планируемого хода с учётом значений исходной целевой функции и содержимого списка исключений. На каждом шаге выбирается такое решение $S' \in N_1(S)$, которое даёт максимальное значение оценочной функции, даже если используется вариант без реального улучшения качества. При выборе ходов без улучшений появляется вероятность возврата к ранее обработанным решениям, наличие списка исключений делает это невозможным.

Для любой реализации поиска с исключениями требуется определить следующее:

1. Запрещающую стратегию, которая задаёт, что именно помещается в список исключений.
2. Освобождающую стратегию, которая задаёт, что должно удаляться из списка исключений.
3. Стратегию, объединяющую поведение двух предыдущих пунктов, включая механизм определения направления поиска и правило выбора следующего решения из $N_1(S)$ на основе стратегии “наилучший подходящий” или “первый подходящий”.
4. Критерий остановки.

В некоторых случаях требуется определение дополнительных стратегий, связанных с особенностями конкретной задачи [50, 51, 75].

1.8.1 Понятие окрестности решения

Прежде всего требуется задать способ определения окрестности решения $N_\lambda(s)$. Автором предлагается это делать при помощи механизма генерации λ -взаимообменов, в котором сначала выбираются два маршрута p и q , затем определяется пара подмножеств их вершин S_p и S_q так, что $|S_p| \leq \lambda$ и $|S_q| \leq \lambda$. Процедура выполняет обмен вершин между S_p и S_q до тех пор, пока это возможно.

Одно из множеств S_p или S_q может быть пустым, в этом случае производится только перенос вершин из одного маршрута в другой. Поскольку количество комбинаций пар, также как и количество вариантов множеств S_p и S_q , может быть достаточно большим, предлагается брать $\lambda = 1$ или $\lambda = 2$.

Эта процедура разработана для симметричных случаев ЗМТ с неопределённым заранее количеством транспортных средств.

1.8.2 Стратегия запрещения

Запрещающая стратегия ограничивает поиск, помечая некоторые ходы как “запрещённые”. Её работа строится на основе запрещающих правил с анализом некоторых характеристик решений. Для избежания зацикливания следует убедиться, что поиск не переходит к решению, которое уже было обработано ранее, но это требует очень больших затрат памяти и процессорного времени. Ниже предлагается специальная структура для хранения некоторых атрибутов решений, заменяющая хранение полного списка исключений.

Структура списка исключений *TABL* имеет вид матрицы размером $(n + 1) \times v$ (n строк, по одной на вершину и v столбцов, по одному для каждого маршрута R_p). Ход может быть представлен двумя парами: (i, R_p) и (j, R_q) , которые обозначают, что вершина i из множества вершин R_p маршрута p была подвергнута обмену с вершиной j из множества вершин R_q маршрута q и наоборот. Атрибуты (i, R_p) и (j, R_q) и используются для определения ограничений, запрещающие некоторый ход. Ход помечается запрещённым, если i возвращается в R_p и j возвращается в R_q . Такой подход удобен тем, что с его помощью можно представить достаточно большое количество решений

при относительно небольших затратах памяти. Это также приводит к меньшему времени выполнения различных операций с списком исключений.

$TABL(i, p)$ хранит номер итерации, на которой вершина i была удалена из множества маршрута R_p . Первоначально матрица $TABL$ заполняется максимальным отрицательным числом, чтобы не допустить ложной идентификации итерации.

1.8.3 Стратегия освобождения удаления из списка исключений

Стратегия освобождения отвечает за порядок удаления из списка исключений решений после ts итераций, где ts обозначает длину списка исключений. Выбор значения ts описывается ниже при помощи функции, зависящей от характеристик задачи и выбора стратегии поиска. Набор запрещённых ходов хранится в списке исключений на протяжении последних ts итераций. Возможность быстрой и простой проверки таблицы очень важна, особенно в случае роста размерности задачи и списка исключений. На некоторой итерации k ход считается разрешённым, если на протяжении ts итераций ни i не возвращается в R_p , ни j не возвращается в R_q . Таким образом, ход запрещён, если:

$$k - TABL(i, p) < ts \text{ и } k - TABL(j, q) < ts. \quad (1)$$

Поскольку $TABL$ хранит номера итераций, факт запрещения хода можно проверить при помощи двух простых проверок на основе формулы (1).

При использовании $TABL$ статус предыдущих ходов обновляется автоматически, в отличие от традиционного циклического списка исключений, в котором требуется дополнительный контроль.

1.8.4 Стратегия выбора

Стратегия выбора — это ключевой момент алгоритма поиска с исключениями. Она предназначена для оценки наилучшего возможного хода из окрестности решений с учётом содержимого списка исключений и так называемого критерия лучшего решения. Ход считается допустимым, если он не запрещён, или запрещён, но прошёл критерий лучшего решения. Запрещающие правила и критерий лучшего решения играют парную роль в ограничении процесса поиска и его направлении [50, 51]. В списке исключений мы храним атрибу-

ты решений, чтобы представить некоторую информацию о них. Некоторые незапрещённые решения могут быть ошибочно заблокированы списком исключений, и критерий лучшего хода позволяет устранить этот недостаток. Сформулируем критерий лучшего решения, который уточняет направление поиска с сохранением уверенности, что не произойдёт заикливания: пусть S_b — текущее наилучшее найденное решение. Если решение $S' \in N_1(S)$ запрещено по причине попадания в список исключений, то будем считать его всё равно допустимым, если $C(S') < C(S_b)$.

Для выбора подходящего хода из списка возможных кандидатов используются две стратегии: “наилучший подходящий” (best-admissible) и “первый наилучший-подходящий” (first-best-admissible). Стратегия “наилучший подходящий” выбирает решение из окрестности, которое даёт наибольшее увеличение целевой функции или наименьшее её ухудшение. Алгоритм поиска с исключениями, использующий стратегию “наилучший подходящий” обозначается TS + BA. Стратегия “первый наилучший-подходящий” является жадным вариантом стратегии “наилучший подходящий”. Она выбирает первое улучшающее решение из окрестности, но в случае, если такого нет, возвращает решение с наименьшим ухудшением. Алгоритм поиска с исключениями на основе этой стратегии обозначается TS + FBA.

1.8.5 Специальная структура данных для стратегии “наилучший подходящий”

Структура данных (СД) для просмотра кандидатов в общих чертах может быть представлена следующим образом: $BSTM$ и $RECM$ — две матрицы размера $v \times v$ и $v(v-1)/2 \times 2$. Верхняя треугольная часть матрицы $BSTM(p, q)$ ($1 \leq p < q \leq v$) используется для хранения изменений в целевой функции Δ_{ij} , связанными с наилучшим ходом при обмене вершины $i \in R_p$ с вершиной $j \in R_q$ или произвольным большим значением, если подобного наилучшего хода не существует. Нижняя треугольная часть $BSTM(q, p)$ используется для хранения положительных индексов l , связанных с парой R_p и R_q во множестве возможных комбинаций пар $\{1, \dots, v(v-1)/2\}$. Каждый индекс указывает положение в матрице $RECM$, где хранятся атрибуты наилучшего хода, например, $RECM(l, 1) = i$ и $RECM(l, 2) = j$.

СД производит вычисление изменений в целевой функции для всех ходов в окрестности решений $N_1(S)$ только один раз на первой итерации. Во время

поиска верхняя часть матрицы $BSTM$ проверяется с целью поиска наилучшего значения Δ_{ij} и определения соответствующих множеств маршрутов. На основе полученной информации и содержимого $RECM$ выбирается следующий ход.

Новый ход затрагивает множества R_p и R_q , все остальные маршруты остаются без изменения. Следовательно, только $2v$ пар маршрутов $(R_p, R_m), \forall p \neq m$, и $(R_q, R_m), \forall q \neq m$, должны быть пересчитаны, вместо полного поиска среди $v(v-1)/2$ пар в случае без описанной оптимизации.

Подобная структура экономит время вычислений без ущерба качеству решений. Алгоритм TS + FBA не может использовать подобную СД, поскольку в нём размер окрестности меняется от итерации к итерации и определяется при выполнении хода.

1.8.6 Функция для определения длины списка исключений

Длина списка исключений ts зависит от характеристик задачи: числа вершин n , числа транспортных средств v , плотности задачи p , которая является отношением потребности клиентов к доступной грузоподъёмности, а также от стратегии выбора (FBA и BA).

Автором алгоритма была получена достаточно удачная эмпирическая оценка путём анализа результатов вычислительного эксперимента. Для случая TS + FBA значение ts может быть оценено как:

$$ts = 8 + (0,078 - 0,067 \times p) \times n \times v.$$

Аналогично, для случая TS + BA оценка выглядит следующим образом:

$$ts = \max\{7, -40 + 9,6 \times \ln(n \times v)\}. \quad (2)$$

1.8.7 Критерий остановки

Критерий остановки в алгоритме поиска с исключениями основан на максимальном количестве итераций ($MAXI$) после того, как было найдено наилучшее решение. Взаимосвязь критерия остановки с изменениями в обрабатываемых решениях имеет очевидные преимущества, приводящие к более эффективному использованию вычислительных ресурсов. Для определения минимального количества итераций M , требуемых для нахождения удачных

решений на основе номеров итераций, где было получено наилучшее решение, были использованы множественные регрессионные тесты. Уравнение для нахождения приближённого значения M было подобрано аналогично ts .

Значение M было подобрано так:

$$M = 340 + 0,000353 \times p \times (n \times v)^2.$$

1.8.8 Общий вид алгоритма

Шаги общего алгоритма поиска с исключениями должны быть следующими:

1. Получаем начальное решение S при помощи алгоритма Кларка-Райта (см. разд. 1.4.1), выполняем цикл поиска для инициализации матриц $BSTM$ и $RECM$ в случае, если выбрана стратегия “наилучший подходящий”. Устанавливаем значение длины списка исключений, подготавливаем список исключений, вычисляем критерий остановки, присваиваем начальное значение счётчику итераций $k = 1$, текущее решение объявляем наилучшим найденным и запоминаем номер итерации, где оно было найдено $k_b = 0$.
2. Выбираем подходящее решение $S' \in N_1(S)$ в соответствии с выбранной стратегией. Записываем атрибуты нового решения в $TABL$. Обновляем текущее решение $S = S'$ и устанавливаем $k = k + 1$. Если стоимость $C(S') < C(S_b)$, обновляем наилучшее решение $S_b = S'$ и номер итерации $k_b = k$. Если используется стратегия “наилучший подходящий”, то также обновляем матрицы $BSTM$ и $RECM$.
3. Если разность $(k - k_b)$ превышает значение, вычисленное для ограничения количества итераций, то переходим на шаг 4. В противном случае, переходим на шаг 2.
4. Останавливаемся, выводим наилучшее решение S_b .

1.9 Другие варианты поиска с исключениями

Помимо алгоритма Османа известен ещё ряд попыток использовать стратегию поиска с исключениями для решения ЗМТ. Можно отметить работы Генро (Gendreau), Герца (Hertz), Лапорте (Laporte) [45], Тейлорда (Taillard) [93],

Ксю (Xu) и Келли (Kelly) [106], Риго (Rego) и Рокарола (Roucairol) [84, 83], а также [17].

1.9.1 Алгоритм Генро-Герца-Лапорте

Алгоритм Генро-Герца-Лапорте, также известный под названием Taburoute [45], основан на предшествующих попытках применения поиска с исключениями для решения задач ЗМТ, но содержит ряд серьёзных улучшений. Структура окрестности решений в этом алгоритме содержит все решения, которые могут быть получены из текущего путём удаления вершины из маршрута, где она находилась, и переносе её в другой, который содержит одного из её p ближайших соседей, используя обобщённую процедуру вставки (GENI — GENeralized Insertion procedure), разработанную Генро, Герцем и Лапорте для решения ЗК [44]. Применение этой процедуры может приводить к удалению существующего маршрута или появлению нового. Другая особенность этого алгоритма заключается в том, что ему позволяет находить решения, нарушающие ограничения грузоподъёмности или длины маршрутов. И тем не менее эти условия не игнорируются полностью. Говоря более точно, здесь целевая функция включает в себя два штрафных компонента: один для штрафования превышений порога грузоподъёмности, а другой — для штрафования превышений времени объезда. Они обновляются каждые десять итераций: делятся на 2, если все десять итераций не нарушали ограничений, или умножаются на 2, если все десять нарушали его.

Алгоритм Генро-Герца-Лапорте использует неявную форму представления списка исключений. В ситуациях, когда вершина перемещена из маршрута r в маршрут s на итерации t , её возвращение в маршрут r запрещено до итерации $t + \theta$, где θ — целое число, случайно выбираемое из интервала от [5, 10]. Алгоритм имеет дополнительный механизм, штрафующий частое перемещение одних и тех же вершин. Он позволяет выбирать такие решения, в которых задействованы давно не используемые вершины. Начальное решение определяется процедурой, которая строит несколько вариантов и выбирает наилучший из них.

Ниже приводится краткое описание алгоритма Генро-Герца-Лапорте. В описании использованы следующие обозначения: W — множество вершин-кандидатов для перемещения в другие маршруты, $q \leq |W|$ — количество вершин из W , с которыми выполнялось пробное перемещение, k — число

смежных итераций, на протяжении которых не было найдено улучшений.

1. Сгенерируем $\lceil \sqrt{n}/2 \rceil$ начальных решений и выполним поиск с исключениями с $w = V \setminus \{v_0\}$, $q = 5$ и $k = 50$. Используемое значение q даёт вероятность выбора одной вершины из каждого маршрута равной по крайней мере 90%.
2. Начиная с наилучшего решения, полученного на шаге 1, выполняем поиск с исключениями при $q = 5$ и $k = 50n$.
3. Используя наилучший результат, найденный на шаге 2, выполним поиск с исключениями при $k = 50$. Здесь W — множество $\lfloor |V|/2 \rfloor$ вершин, которые наиболее часто перемещались на шагах 1 и 2. При поиске q выбирается равным $|W|$.

Описанный алгоритм на практике оказывается одним из самых эффективных по стоимости решений, но при этом его результаты сложно сравнивать с другими алгоритмами, т. к. он использует систему мягких ограничений при штрафовании вместо более традиционной жёсткой.

1.9.2 Алгоритм Тейлорда

Алгоритм Тейлорда (Taillard) [93] включает в себя некоторые механизмы, использованные в алгоритме Генро-Герца-Лапорте. К их числу принадлежит, например, случайное определение времени хранения элементов в списке исключений. Понятие окрестности решений в алгоритме Тейлорда определяется при помощи механизма генерации λ -взаимообменов, который был предложен Османом (см. п. 1.8.1). Алгоритм использует простой механизм добавления вершин, вместо обобщённой процедуры вставки (см. п. 1.9.1). Это позволяет выполнять вставку за меньшее время, а также никогда не приводит к нарушению наложенных ограничений. В ходе работы алгоритма часто запускается процедура оптимизации отдельных маршрутов на основе метода Волгенанта-Джонкера (Volgenant and Jonker) [102].

Новшество алгоритма Тейлорда заключается в том, что он позволяет проводить декомпозицию задачи на подзадачи. В задачах на плоскости декомпозиция проводится путём разделения вершин на сектора с центром разбиения, расположенным в депо. Каждая подзадача решается независимо, но периодически возникает необходимость переноса вершин в соседние сектора. Для

задач не на плоскости автор предлагает специальный метод декомпозиции. Возможность декомпозиции — это очень важная особенность, т. к. позволяет эффективно применять этот метод на многопроцессорных системах.

1.9.3 Алгоритм Ксю-Келли

Алгоритм Ксю-Келли (Xu and Kelly) [106] использует более сложное определение понятия окрестности решений. В него входит обмен вершин между двумя маршрутами, глобальное перепозиционирование вершины в другие маршруты и локальная оптимизация одного маршрута. Стратегия глобального перепозиционирования вершины использует модель потоков в сетях для поиска оптимального перемещения заданного количества вершин в другие маршруты. Разработаны приближительные методики оценки стоимости извлечения вершины из одного маршрута и вставки вершины в другой с учетом грузоподъемности транспортных средств. Оптимизация отдельного маршрута проводится при помощи метода 3-опт (см. п. 1.6.1). Алгоритм управляется несколькими параметрами, которые подбираются в процессе поиска. В ходе работы ведётся специальный пул решений для перезапуска процесса от некоторой точки с изменёнными значениями параметров. Этот алгоритм иногда находит очень удачные решения, но его применение затруднено, т. к. он требует очень тщательного подбора параметров, что не всегда легко выполнить на практике.

1.9.4 Алгоритм Риго-Рокарола

Главная особенность алгоритма Риго-Рокарола (Rego and Roucairol) [84]) заключается в том, что он использует списки извлечений (ejection chains) для перехода от одного решения к другому. Извлечение заключается в перемещении вершины на место, занимаемое другой вершиной. Это приводит к созданию “цепной реакции” (chain reaction) l уровней.

Для некоторого направления маршрута, определяемого вершиной v_{i-1} , предшествующей вершине v_i , и вершиной v_{i+1} , следующей за v_i , цепочка извлечений l уровня состоит из замен троек $(v_{i-1}^k, v_i^k, v_{i+1}^k)$ ($k = 0, \dots, l$) тройками $(v_{i-1}^k, v_i^{k-1}, v_{i+1}^k)$ ($k = 1, \dots, l$) и перемещении v_i^l . В алгоритме присутствует дополнительное ограничение допустимости операции, которое позволяет убедиться, что новое решение корректно. Например, оно проверяет, что никакая

дуга не встречается больше одного раза. На начальном шаге работы алгоритма определённое число вершин выбираются как кандидаты для извлечения вместе с их непосредственными соседями при соблюдении ограничения допустимости операции. Как и в алгоритме Генро-Герца-Лапорте, в этом методе допускается получение решений с нарушением ограничений грузоподъёмности или общей длины маршрута.

Результаты работы этого алгоритма в среднем не достигают качества наиболее удачных применений поиска с исключениями. Известны попытки его модификации [83], но они также не позволили приблизиться к самым удачным вариантам.

1.10 Моделируемый отжиг

Метод моделируемого (имитируемого) отжига (Simulated Annealing) — это алгоритм, исследование которого велось с целью получения эвристики, способной к преодолению локальных экстремумов целевой функции в процессе поиска. Он способен продолжить работу, что в некоторых случаях даёт возможность найти глобальный экстремум. ЗМТ— это не единственная задача, которая может быть решена при помощи метода моделируемого отжига. Известны также и другие приложения этого алгоритма в сфере дискретной оптимизации.

Термин “моделируемый отжиг” взят из физики. Идея метода появилась при наблюдении процессов, происходящих в металле в ходе охлаждения после достижения им температуры плавления. Известно несколько вариантов моделируемого отжига для решения ЗМТ. Наиболее удачным из них является алгоритм Османа, который предложил также свой подход и к поиску с исключениями, описанный в разд. 1.8.

Во всех вариантах алгоритма используется понятие окрестности решений. В окрестность входят все решения, которые могут быть получены путём выполнения фиксированного количества элементарных преобразований из текущего решения. Набор преобразований в каждом варианте алгоритма различен и описывается отдельно.

На каждой итерации работы алгоритма t решение x выбирается случайно из окрестности $N(x_t)$, где x_t — последнее решение, найденное на предыдущих итерациях [43]. Если стоимость нового решения $f(x) < f(x_t)$, тогда x_{t+1} устанавливается равным x , в противном случае выполняется одно из двух

действий:

$$x_{t+1} = \begin{cases} x & \text{с вероятностью } p_t \\ x_t & \text{с вероятностью } 1 - p_t, \end{cases}$$

где p_t чаще всего выбирается как бывающая функция от t и от разности $f(x) - f(x_t)$. Наиболее распространённое определение p_t выглядит следующим образом:

$$p_t = \exp(-[f(x) - f(x_t)]/\theta_t), \quad (3)$$

где θ_t обозначает “температуру” на итерации t . Правило, используемое для определения θ_t , принято называть “охлаждающим расписанием” (cooling schedule). В большинстве случаев θ_t выбирается как убывающая дискретная функция от t . Её начальное значение задаётся равным $\theta_1 > 0$ и умножается после каждой итерации T на коэффициент α ($0 < \alpha < 1$). Это приводит к уменьшению вероятности принятия наихудшего решения .

1.10.1 Ранние алгоритмы моделируемого отжига для решения ЗМТ

В [86] представлены два ранних алгоритма моделируемого отжига для решения ЗМТ. В первом варианте авторы определяли понятие окрестности решений путём сочетания нескольких преобразований: разворот части маршрута, перемещение части маршрута на другое место того же маршрута, обмен вершинами между маршрутами. В другом варианте [14] для построения начального решения используется двухфазовый алгоритм с предшествованием решения ЗК перед кластеризацией (см. разд. 1.5.5) и использованием метода 3-опт (см. п. 1.6.1) для дальнейших улучшений.

1.10.2 Алгоритм Османа

Реализация алгоритма моделируемого отжига, предложенная Османом [74], значительно более удачна, чем варианты, упомянутые в предыдущей главе. В ней используется более глубокий метод поиска начального решения, реализован подбор некоторых параметров во время пробной фазы работы алгоритма, ведётся более глубокий поиск соседних решений, а также применяется глубже продуманное охлаждающее расписание.

Понятие окрестности решений в этом алгоритме задаётся таким же образом, как и в алгоритме Османа поиска с исключениями (см. п. 1.8.1). Опишем первую фазу работы (подготовительный этап):

1. Построим начальное решение при помощи алгоритма Кларка-Райта (см. разд. 1.4.1);
2. Выполним проверку соседних решений при помощи схемы λ -взаимообменов и внесём улучшения, если они были найдены. Остановим работу, если в окрестности решений нет больше подходящих вариантов обмена.

Вторая фаза работы (реализация моделируемого отжига) выглядит следующим образом:

1. Исследуем решение, полученное в конце первой фазы работы алгоритма. Применяем ещё раз схему λ -взаимообменов (без внесения изменений), вычисляем Δ_{max} и Δ_{min} — наибольшее и наименьшее возможные изменения целевой функции, зафиксированные во время просмотра, и β — количество возможных обменов. Установим $\theta_1 = \Delta_{max}$, $\delta = 0$, $k = 1$, $k_3 = 3$, $t = 1$, $t^* = 1$ (это итерация, на которой было найдено наилучшее решение). Пусть x_1 — текущее решение, а $x^* = x_1$ — самое лучшее решение, найденное на текущий момент.
2. Проверим окрестность решений для x_t , используя схему λ -взаимообменов. Если решение x с $f(x) < f(x_t)$ найдено, установим $x_{t+1} = x$. Если $f(x) < f(x^*)$, установим $x^* = x$ и $\theta^* = \theta_k$. Если полный поиск улучшений больше не даёт результатов лучше, чем x_t , тогда пусть x будет лучшим решением в окрестности x_t , и установим

$$x_{t+1} = \begin{cases} x & \text{с вероятностью } p_t \\ x_t & \text{с вероятностью } 1 - p_t, \end{cases}$$

где p_t определяется как (3). Если $x_{t+1} = x_t$, установим $\delta = 1$.

3. Если $\delta = 1$, установим $\theta_{t+1} = \max\{\theta_t/2, \theta^*\}$, $\delta = 0$ и $k = k + 1$. Если $\delta = 0$, установим $\theta_{t+1} = \theta_t / [(n\beta + n\sqrt{t})\Delta_{max}\Delta_{min}]$. Установим $t = t + 1$. Если $k = k_3$, то останавливаемся. В противном случае переходим на шаг 2.

Охлаждающее расписание, использованное Османом, отличается от того, которое обычно применяется в алгоритмах моделируемого отжига. Температура уменьшается ни как непрерывная функция, ни как дискретная. Здесь она уменьшается непрерывно, пока идут изменения решения, но в случаях, когда $x_{t+1} = x_t$, текущая температура либо уменьшается вдвое, либо заменяется температурой, при которой было найдено улучшение решения.

1.11 Детерминированный отжиг

Детерминированный отжиг (Deterministic Annealing) работает таким же образом, как и моделируемый отжиг, за исключением того, что для принятия решения о следующем шаге используется явно детерминированный механизм. Известны два подхода детерминированного отжига:

1. Пороговое принятие (Threshold accepting) [36].
2. Ход от рекорда к рекорду (Record to record travel) [35].

В алгоритме порогового принятия на некоторой итерации t решение x_{t+1} принимается, если $f(x_{t+1}) < f(x_t) + \theta_1$, где θ_1 — параметр, указываемый пользователем.

В алгоритме хода от рекорда к рекорду рекордом называется наилучшее решение x^* , зафиксированное в процессе поиска. На некоторой итерации t решение x_{t+1} принимается, если $f(x_{t+1}) < \theta_2 f(x_t)$, где θ_2 — параметр, указываемый пользователем и обычно имеющий значение немногим больше единицы.

1.12 Генетический алгоритм

Генетический алгоритм (genetic algorithm) — методика решения некоторых задач путём имитации процессов, наблюдаемых в ходе эволюции естественной природы. Впервые парадигма генетического алгоритма была предложена в [58], но понадобилось ещё почти десять лет, чтобы на неё обратили внимание в широких научных и исследовательских кругах. В чистом виде генетический алгоритм — это общая методика решения задач, для реализации которой требуется относительно небольшое количество информации о предметной области. Как следствие этот подход может быть использован для достаточно большого числа плохо структурированных задач [60], где специализированные методы не показывают удачных результатов.

В своей работе генетический алгоритм пытается развивать популяцию битовых строк (bitstrings) или “хромосом”, где каждая хромосома кодирует некоторое решение задачи в некоторой её конкретной постановке. Такая эволюция реализуется путём применения нескольких операторов, имитирующих феномены живой природы (воспроизведение, видоизменение и. т. д.). Ниже приводится общее описание генетического алгоритма, а затем будет описано, как применить подобную стратегию к ЗМТ.

1.12.1 Основной вид генетического алгоритма

Опишем основную схему работы генетического алгоритма. Он начинает выполнение с некоторой начальной популяции хромосом $X^1 = \{x_1^1, \dots, x_N^1\}$, полученной при помощи датчика случайных чисел. На каждой итерации $t = 1, \dots, T$ выполняются k раз шаги с 1 по 3, приведённые ниже ($k \leq N/2$), затем выполняется шаг 4.

1. Выбираем две родительские хромосомы из X^t .
2. Порождаем два потомка на основе двух родительских хромосом при помощи оператора скрещивания (crossover operator).
3. Выполняем операцию мутации для каждого потомка (с малой вероятностью).
4. Создаём X^{t+1} из X^t путём удаления $2k$ наихудших решений в X^t и замены их $2k$ новых потомков.

В приведённом алгоритме параметр T — количество поколений, и k — количество выборок в поколении. Наилучшее решение, полученное за T поколений, является окончательным результатом работы алгоритма. На шаге 1 выбор родительских хромосом вероятностно склоняется к выбору наилучших вариантов. На шаге 2 потомок генерируется скрещиванием путем сочетания битовых строк, найденных у родителей. Каждый потомок может быть затем немного модифицирован на шаге 3 путём изменения значения битов в битовой строке с малой вероятностью для каждой битовой позиции. Ожидается, что первоначальное случайно построенное множество будет улучшено в ходе описанного процесса. Такого вывода придерживаются некоторые теоретические исследования [58, 54].

1.12.2 Применение генетического алгоритма для задач упорядочивания

Классический подход генетического алгоритма, представленный ранее, не подходит для задач упорядочивания (sequencing problems), к которым относятся ЗК и ЗМТ. Прежде всего, для таких задач битовое представление решения не является естественным. Например, его можно заменить представлением путём использования цепочек чисел, где каждое число обозначает некоторую вершину. Положение каждого числа в строке представляет положение вершины в маршруте, предполагая, что последняя вершина замкнута с первой. Далее, должны быть разработаны новые специальные операторы скрещивания и мутации для получения потомков, которые учитывали бы порядок элементов.

Прямое применение классического алгоритма приводит и к другим подобным проблемам. Известны предложения специализированных вариантов операторов скрещивания и мутации для получения цепочек потомков на основе родительских решений [79]. Одно из них — предложение оператора “упорядоченное скрещивание” [72]. Сначала случайно выбираются две вершины для вырезания, и подстрока, находящаяся между ними, приписывается потомку. Оставшиеся позиции заполняются вершинами, начиная со второй выбранной, в таком порядке, как они встречаются во втором родительском решении, соблюдая циклическую замкнутость маршрута и отслеживая повторения. Вторым потомком может быть получен тем же способом при помощи обмена роли родительских решений. В этом алгоритме потомок обычно наследует относительный порядок вершин от родительских строк. Другие операторы стараются сохранять порядок вершин [55] или порядок дуг родительских решений [103].

1.12.3 Применение алгоритма для решения ЗМТ

Литература по разработке алгоритмов для решения ЗМТ на основе генетического алгоритма довольно скудна в отличие от его применения для решения ЗК [79] или от более сложных вариантов ЗМТ, где учитываются временные окна или есть ограничения предшествования [21, 81, 80, 96, 94, 97]. Отставание по сравнению с ЗК объясняется тем, что ЗК — это хорошо изученная каноническая комбинаторная задача, хорошо подходящая для исследования

и пробного применения новых идей вообще и генетического алгоритма в частности, а сложные ограничения тяжело обрабатывать общими алгоритмами. Генетический алгоритм хорошо подошёл для случаев со сложными ограничениями, т. к. он требует относительно небольшое количество информации о природе самой задачи. В настоящий момент можно говорить, что генетический алгоритм имеет хороший потенциал для создания быстрых методов поиска решений ЗМТ в сочетании с обработкой сложных видов ограничений. Это подтверждают существующие предложенные методы применения генетического алгоритма для решения ЗМТ в окнах времени. Выполнена работа в области решения ЗМТ с ограничениями по грузоподъёмности, где уделялось особое внимание влиянию различных параметров на качество получаемых результатов.

В ЗМТ решение содержит несколько маршрутов (в отличии от ЗК), поэтому представления цепочек для генетического алгоритма расширено и депо включается в них многократно. Каждое включение депо служит разделителем между двумя соседними маршрутами.

Применяемый классический оператор скрещивания, называемый PMX [55] и оператор мутации RAR были адаптированы для использования в новых условиях. На каждой итерации они применяются до тех пор, пока не будет получено требуемое количество решений, удовлетворяющих ограничениям, а остальные потомки отбрасываются. Автор применял специальный оператор локального спуска (local descent operator), основанный на четырёх различных типов изменения порядка выполняемых шагов, и вызывал его только для наилучших решений в текущей популяции. В работе показано, что применение оператора локального спуска даёт заметное улучшение производительности алгоритма. В целом, наилучшие решения, получаемые при помощи генетического алгоритма, моделируемого отжига и поиска с исключениями, сравнимы по качеству. Генетический алгоритм требует несколько больше процессорного времени для работы, чем два других подхода. Пока не приведено сравнительных результатов качества работы генетического алгоритма относительно алгоритма на основе муравьиных колоний и нейронных сетей.

Интересный вариант генетического алгоритма приведён в [88, 89]. Он может быть использован в задачах с ограничениями на окна времени. Интересная особенность этого алгоритма в том, что в нём используется классический подход с разрезанием общего маршрута, полученным после решения

ЗК на общем множестве вершин (см. разд. 1.5.5), что позволяет использовать традиционную схему представления решения без разделителей в виде многократного включения депо. Для разделения на маршруты каждого экипажа применяется процедура заметания (см. разд. 1.5.1), начиная работу с первой вершины, находящейся на первом месте в генетической цепочке. Переход к следующему маршруту происходит, когда превышает ограничение на длину маршрута или грузоподъёмности. Таким образом генетическая цепочка может быть всегда преобразована в допустимое решение ЗМТ.

Реализация алгоритма, предложенная в [89], использует оператор скрещивания ОХ и оператор мутации, который основан на обмене местами двух случайно выбранных вершин. Для повышения качества работы этот оператор в дальнейшем был заменён на использование алгоритма 2-опт (см. п. 1.6.1), запускаемый для каждой вершины с вероятностью 0,15.

В [18] предложена схема кодирования, основанная на случайных ключах. В такой схеме каждый элемент задачи связывается со случайно сгенерированным ключом. Преобразование таких ключей в решение задачи можно выполнить при помощи операции сортировки. Непосредственно в ЗМТ каждая вершина-клиент связывается со случайным целым числом, обозначающим транспортное средство, которое будет обслуживать эту вершину, плюс число с плавающей точкой в интервале $(0, 1)$. При сортировке этих чисел можно получить последовательности вершин для каждого маршрута. Подобное применение случайных чисел для решения ЗМТ приводится скорее для иллюстрации такой возможности, но оно не исследовалось достаточно детально.

1.13 Алгоритм на основе муравьиных колоний

Муравьиные колонии (Ant Systems) — ещё одно явление природы, которое послужило источником новой идеи для решения задач комбинаторной оптимизации. Идея родилась в ходе наблюдения процесса поиска пищи в колониях муравьёв. В своём поиске муравьи отмечают пройденный путь, выбрасывая специальные ароматические эссенции, называемые феромонами. Количество феромонов зависит от длины пути и от качества пищи, найденной муравьями. Оставленные следы дают информацию остальным муравьям, привлекая их запахом. Со временем, муравьи всё чаще проходят пути, ведущие к источнику пищи или близкие к гнезду колонии, таким образом на них остаётся всё больше феромонов. Муравьи выработали очень эффективную процедуру

поиска пищи при помощи описанного механизма.

В [28] была представлена новая стратегия решения задач комбинаторной оптимизации. В ней воображаемые “муравьи” исследуют пространство решений, имитируя процесс поиска пищи естественными муравьями. Значение целевой функции сопоставляется с понятием качества пищи. Чтобы проиллюстрировать основные принципы алгоритма на основе муравьиных колоний, приведём описание подхода для решения ЗК. С каждой дугой (v_i, v_j) ассоциируются два значения: видимость n_{ij} (обратно пропорционально длине дуги), которая является статическим параметром, и след феромонов — Γ_{ij} , который обновляется динамически в ходе работы алгоритма. На каждой итерации воображаемые муравьи выполняют построение новых n маршрутов, начиная из всех вершин графа и используя эвристику вероятного ближайшего соседа с модифицированным измерением расстояния. Такое измерение учитывает величину n_{ij} и Γ_{ij} , чтобы вести поиск ближайших соседей, доступных через рёбра с высоким значением феромонов. Предполагается, что некоторая часть феромонов испаряется. Для этого введён дополнительный коэффициент $0 \leq \rho \leq 1$. В конце каждой итерации перед тем, как обновить значения феромонов, из их старой величины вычитается доля $(1 - \rho)$. Если ребро (v_i, v_j) было использовано муравьём k и длина маршрута, построенного этим муравьём, равна L_k , то след феромонов увеличивается на $\Delta_{ij}^k = 1/L_k$. Значение отметки для (v_i, v_j) обновляется таким образом:

$$\Gamma_{ij} = \rho\Gamma_{ij} + \sum_{k=1}^N \Delta_{ij}^k,$$

где N — количество муравьёв. Приведённый процесс построения маршрутов и обновлений феромонов выполняется фиксированное количество итераций. Коэффициент испарений $(1 - \rho)$ предотвращает получение плохих решений на ранних итерациях.

Общие заключения, которые могут быть сделаны на основе существующего опыта [29, 30, 32, 33, 41], сводятся к тому, что иногда этот метод даёт отличные результаты, но в целом он всё же не может соревноваться с другими метаэвристическими или специализированными алгоритмами, если не совмещать его с каким-нибудь видом дополнительных локальных оптимизаций.

Известны только три работы, посвященные применению алгоритма на основе муравьиных колоний для решения ЗМТ. Одна из них [63] предлага-

ет комплексный подход к использованию муравьиных колоний в сочетании с оптимизацией 2-опт и применением вероятностного подхода принятия решений, напоминающего моделируемый отжиг. В работе приведены только результаты запуска этого алгоритма на двух вариантах задачи: с 30 и 60 вершинами, в которых он обнаружил оптимальные решения. Больше не проводилось тестирования этого метода, что не позволяет делать более глубоких выводов о его эффективности. В двух других работах [24, 25] описывается новая идея алгоритма и её улучшение. В первом варианте метода авторы совместили алгоритм муравьиных колоний с оптимизацией 2-опт каждого маршрута на каждой итерации до выполнения процедуры обновления путей феромонов. Этот алгоритм учитывает понятия грузоподъёмности транспортного средства и другие ограничения при выборе следующей вершины для обхода муравьём. Во втором варианте авторы пересмотрели свой алгоритм в нескольких направлениях:

1. Понятие грузоподъёмности, ранее использованное в процессе выбора вершины, опущено, т. к. это приводило к дорогим вычислениям, и на его место было введено понятие сбережения, встроенное в понятие видимости, в параметрическом виде.
2. Только $\lfloor n/4 \rfloor$ вершин выбираются для дальнейшего посещения.
3. Только 5 наилучших решений выбираются на каждой итерации для обновления отметок, и качество наносимых феромонов варьируется в зависимости от ранга решения.

Приведённые изменения позволили уменьшить время вычислений и улучшить результаты. В целом можно говорить, что алгоритмы муравьиных колоний пока мало исследованы для применения в решении ЗМТ. В этом направлении есть большое количество возможных вариантов для дальнейших исследований.

1.14 Нейронные сети

Нейронные сети (Neural networks) — это вычислительная модель, состоящая из элементов, сильно связанными множеством соединений, напоминающими нейроны в головном мозге. Для каждого соединения назначается весовой коэффициент. От одного элемента может быть послан сигнал другому, который

передаётся через соответствующее соединение с учётом весового коэффициента. При сопоставлении с биологическими двойниками, воображаемые нейронные сети обнаруживают особенности, присущие человеческому восприятию. В частности, они способны обучаться, накапливать опыт и приобретать общие представления на основе некоторых примеров путём постепенного подгона весовых коэффициентов. Изначально эта модель была разработана для задач, связанных с человеческим интеллектом, где традиционные методы вычислений оказались несостоятельными, например, при распознавании речи. С некоторого момента они стали применяться для решения комбинаторных задач. Начало положила работа [59]. Применение нейронных сетей для решения ЗК стало предметом многих исследований, таких как метод эластичных сетей (Elastic Net) [37] и метод самоорганизующейся карты (Self-Organizing Map) [65]. Эти методы были достаточно далеки от классических подходов к построению нейронных сетей, но давали результаты лучше, чем попытки применить нейронные сети в их чистом виде. И тем не менее они уступают по своему качеству и не могут соревноваться с другими метаэвристическими алгоритмами.

Эластичные сети и самоорганизующиеся карты представляют собой шаблоны с возможностью деформации, которые подгоняют себя к контурам вершин, чтобы решить ЗК. Элементы соединяются так, чтобы построить замкнутое кольцо. Начиная с некоторой случайной конфигурации, положение каждого элемента в кольце постепенно подгоняется (наподобии весовых коэффициентов в связях в обычных нейронных сетях) до тех пор, пока не будет достигнуто некоторое приемлемое расстояние от элементов до вершин. Процесс завершается назначением каждой вершине ближайшего элементу модели. Посредством произведённого назначения порядок элементов на кольце определяет порядок вершин в решении задачи.

Эластичные сети и самоорганизующиеся карты очень похожи, но отличаются подходом к контролю подгонки элементов к вершинам. Деформирующиеся шаблоны очень хорошо подходят к решению геометрических задач, например, евклидовой ЗК. При этом описанные подходы не имеют возможности для отслеживания нарушений наложенных ограничений, как, например, ограничений по грузоподъёмности. Такие ограничения сильно нарушают геометрическую природу задач.

Пока только небольшое количество попыток посвящено поиску применения

нейронных сетей для решения ЗМТ. В основном они опирались на самоорганизующиеся карты [46, 47, 48, 71, 90]. Обобщение моделей, применяемых для решения ЗК, к решению ЗМТ заключается в применении большого количества деформирующихся шаблонов, по одному на каждый маршрут. Обычно модель применяется с использованием постоянно увеличивающегося числа колец (маршрутов) до тех пор, пока не будет найдено подходящее решение. По причине существования нескольких колец между ними появляется соперничество при разделении вершин.

Алгоритм, представленный в [47], для ЗМТ с учётом грузоподъёмности в основных чертах может быть представлен следующим образом:

1. Повторяем приведённые ниже шаги до тех пор, пока есть элементы достаточно близкие к каждой вершине:
 - (a) перемещаемся по кругу после обработки последней вершины и выбираем следующую вершину как текущую;
 - (b) установим каждому кольцу вероятность его выборки;
 - (c) выберем кольцо в соответствии с вероятностью выборки, установленной на предыдущем шаге;
 - (d) экспериментально подберём для текущей вершины ближайший элемент на выбранном кольце, и будем его перемещать по направлению к текущей вершине (вместе с соседними элементами).
2. Окончательно сопоставим вершины и элементы колец для получения решения.

Вероятность, ассоциированная с каждым кольцом, постоянно уточняется по мере того, как идёт работа алгоритма. В начале работы алгоритма расстояние между текущей вершиной и выбранным элементом на кольце имеет преимущественное значение. Далее в ходе работы больше влияния должны оказывать ограничения грузоподъёмности, чтобы назначение вершины без нарушений ограничений стало бы всё менее и менее вероятным (в силу пробного выбора элемента). Постепенно кольцам становится сложнее точно приспособиться к вершинам. В конце работы могут быть выбраны только кольца, не приводящие к нарушениям ограничений. В последующих публикациях этот алгоритм был приспособлен также для обработки ограничений

максимальной длины маршрута при помощи модификации правил вычисления вероятности выбора колец [48]. В целом, на вычислительном эксперименте этот алгоритм показывает интересные результаты, но не может сравниться с другими развитыми алгоритмами. В частности, по качеству решений поиск с исключениями даёт результаты лучше.

1.15 Выводы

Задача маршрутизации транспорта (ЗМТ) может быть представлена следующим образом:

1. Требуется построить несколько замкнутых маршрутов, проходящих через заданное множество целевых вершин. Через каждую вершину должен проходить только один маршрут.
2. Все маршруты должны проходить через депо.
3. Общая стоимость объезда маршрутов должна быть минимальна.

Кроме основного вида, важными являются следующие разновидности с дополнительными ограничениями [95, 77, 15]:

1. ЗМТ с учётом грузоподъёмности.
2. ЗМТ с ограничением количества вершин в маршрутах.
3. ЗМТ для нескольких депо.

Точные методы решения ЗМТ не применяются из-за чрезмерно большого времени вычислений. Классификация приближённых методов выглядит следующим образом:

- Классические эвристики.
 1. Конструктивные алгоритмы.
 2. Двухфазные алгоритмы:
 - (а) сначала разделение вершин на группы, затем решение ЗК;
 - (б) сначала решение ЗК, затем разрезание маршрута на фрагменты.
 3. Улучшающие алгоритмы.
- Метаэвристики.

1. Поиск с исключениями.
2. Моделируемый и детерминированный отжиг.
3. Генетический алгоритм.
4. Алгоритм на основе муравьиных колоний.
5. Нейронные сети.

Классические эвристики заложили основные подходы к решению ЗК. В настоящее время вытеснены метаэвристиками, позволяющими получать качественные решения, но с предварительной процедурой вариации управляющих параметров, содержащихся в их описании. Необходимость выполнения дополнительной настройки для каждого нового набора данных является серьёзным недостатком метаэвристик, затрудняющим их внедрение в программные пакеты для массового применения.

2 Сбалансированные алгоритмы решения ЗМТ

В этой главе предлагается новый подход к решению ЗМТ, основанный на использовании понятия сбалансирования уровня загрузки транспортных средств. Ниже последовательно рассматриваются применения идеи сбалансирования для следующих четырёх видов ЗМТ:

1. Сбалансированной ЗМТ.
2. ЗМТ с учётом грузоподъёмности.
3. ЗМТ с учётом грузоподъёмности и с ограничением количества вершин в каждой группе.
4. ЗМТ для нескольких депо.

Три последних вида ЗМТ являются широко известными вариантами этой задачи. Для них условие сбалансирования включается в реализацию соответствующих алгоритмов, но не отражено в постановке, и, как следствие, полученные решения не обладают никакими формальными особенностями. Сбалансированная ЗМТ— это новая постановка, в которой условие сбалансирования задано в явной форме.

Насколько можно судить, существует очень мало попыток применить принцип сбалансирования при решении ЗМТ. Например, в работе [70] описан подход идейно близкий, но отличный по формальным критериям. Авторы предлагают введение дополнительного ограничения — ограничения загрузки транспортного средства снизу, что отличается от формы, предлагаемой в этой работе.

Идея сбалансирования может быть интересна как с точки зрения прямой практической пользы, т. к. выравнивает загрузку транспортных средств, предотвращая их нерациональное использование, так и с точки зрения повышения скорости работы алгоритмов. Как можно было заметить в предыдущей обзорной главе, подавляющее большинство алгоритмов, способных находить качественные решения, выполняют очень глубокий поиск с просмотром большого количества различных комбинаций. В частности, генетический алгоритм обрабатывает сразу некоторый массив решений — популяцию, от размера которой зависит качество результата. Чем популяция больше, тем больше вероятность попадания в неё удачного решения. Поиск с исключениями

и моделируемый отжиг обрабатывают в каждый момент времени только одно решение, но стараются как можно дальше отдалить момент прекращения поиска, используя механизмы преодоления точки локального минимума.

Предлагаемое условие сбалансирования ограничивает пространство поиска, тем самым сокращая время работы алгоритмов. Задача может быть разбита на две части:

1. Построить новые алгоритмы с использованием условия сбалансирования.
2. Оценить влияние условия сбалансирования на качество решений, получаемых новыми алгоритмами.

Первая часть задачи решается при помощи разработки метода дихотомического деления вершин на группы для каждого транспортного средства (допустимым является синонимичное название “метод или процедура дихотомической кластеризации”). Он выступает в роли некоторой алгоритмической схемы, позволяющей строить конкретные новые алгоритмы приближённого решения той или иной разновидности ЗМТ. Вторая часть задачи решается при помощи проведения вычислительного эксперимента, в котором сбалансированный алгоритм сравнивается с алгоритмом Османа поиска с исключениями. Алгоритм Османа поиска с исключениями является очень распространённой метаэвристикой, показывающей достаточно удачные результаты среди алгоритмов, применяемых для решения ЗМТУГ в точной формулировке. Известны алгоритмы, способные найти маршруты, более дешёвые, чем маршруты, полученные алгоритмом Османа, как например алгоритм Генро-Герца-Лапорте (см. гл. 1.9.1), но они допускают нарушение ограничения грузоподъёмности с введением системы штрафов, принятого недопустимым в рамках требуемого эксперимента.

Порядок выполнения процедуры дихотомического деления вершин на группы неразрывно связан с условием сбалансирования и фактически возможен только при его наличии. Условие сбалансирования позволяет производить планирование уровня загрузки транспортных средств, необходимого для предложенного метода.

Введение сбалансированного вида ЗМТ позволило выбрать наилучший вариант дихотомического деления вершин на группы, заранее отбросив идеи, негативно влияющие на результат. Таким образом, она помимо практической ценности несёт и некоторую теоретическую значимость при исследовании.

Полезность процедуры дихотомического деления не ограничивается только рамками рассмотренных видов ЗМТ. Исследования в этом направлении могут быть продолжены в будущем с получением новых алгоритмов для других постановок задачи.

Далее мы рассмотрим точную формулировку сбалансированной задачи (разд. 2.1), дихотомический алгоритм для решения задач без учёта грузоподъёмности транспортных средств (разд. 2.4), две модификации известных алгоритмов для сбалансированной задачи (разд. 2.8 и разд. 2.7), дихотомический алгоритм для задач с учётом грузоподъёмности (разд. 2.10), а также уделим внимание модификации алгоритмов для случая с несколькими депо (разд. 2.14).

2.1 Сбалансированная ЗМТ

Предлагаемая в этом разделе постановка задачи введена для случаев, когда допускается наличие жёсткого формального ограничения сбалансированности количества вершин для каждого транспортного средства. Среди особенностей этой формулировки можно отметить то, что в ней количество транспортных средств задаётся явно, а потребность в товаре вершин-клиентов либо неизвестна, либо может быть принята для всех одинаковой.

На практике предложенная формулировка может применяться, например, для вычисления маршрутов почтальонов, разносящих денежные выплаты, или для планирования поездок курьеров, доставляющих различные документы. В случае денежных выплат сумма, которую почтальон способен брать с собой, может считаться неограниченной, в то время как равномерность количества адресов для каждого из них становится значительно более важным критерием. Обход маршрута каждого почтальона должен по времени укладываться в рамки продолжительности рабочего дня. Сбалансированная задача не допустит появления непредвиденно длинных списков точек назначения, а уровень их наполнения можно регулировать за счёт выбора оптимального количества маршрутов.

С другой стороны, применение сбалансированного вида ЗМТ может быть эффективным только в случае, когда вершины расположены относительно равномерно, если рассуждать в понятиях плоского пространства. Если же вершины сгруппированы в несколько достаточно отдалённых друг от друга групп, особенно в случае различного количества вершин в каждой группе,

сбалансированный алгоритм породит решения, захватывающие по несколько локально обособленных частей, что, очевидно, будет крайне неэффективно. Подобные варианты лучше обрабатывать специализированными методами, либо вручную разбить общую задачу на несколько подзадач, каждую из которых решить в отдельности.

Идея сбалансированной загрузки должна требовать равное количество вершин в каждом маршруте, но абсолютно равным оно может быть только в случае, когда количество целевых вершин кратно количеству транспортных средств, что обычно не выполняется. Поэтому условие было несколько модифицировано и требует *почти* равного количества, т. е. с ограничением отличаться не более, чем на единицу.

В точном виде сбалансированная задача маршрутизации транспорта (СЗМТ) может быть представлена следующим образом:

1. Пусть дано множество вершин $V = \{v_0, v_1, \dots, v_n\}$ из $n + 1$ элементов.
2. Пусть $V' = \{v_1, v_2, \dots, v_n\}$ множество из n целевых вершин, а v_0 — вершина-депо.
3. Пусть задана симметричная матрица стоимости переездов между всеми $n + 1$ вершинами из V , в которой для любого $i \in V C_{ii} = 0$ и для любых $i, j \in V V_{ij} \geq 0$.
4. Пусть k — количество транспортных средств, заданное явно.
5. Необходимо построить k замкнутых маршрутов минимальной суммарной стоимости на основе значений матрицы C , таких что вершина v_0 включается во все из них, а через каждую вершину из V' может пройти только один из построенных маршрутов.
6. Количество вершин в каждой паре полученных маршрутов не может отличаться более, чем на единицу.

Как и простая ЗМТ СЗМТ продолжает оставаться NP-трудной, поэтому мы будем рассматривать только приближённые методы её решения.

2.2 Вычисление количества вершин для каждого транспортного средства в СЗМТ

Одной из главных особенностей, отличающих СЗМТ от всех прочих видов ЗМТ, является возможность заранее вычислить количество вершин для по-

сещения каждым транспортным средством. Как было определено в разд. 2.1, длины маршрутов для любой пары транспортных средств не должны отличаться более чем на одну вершину. Таким образом, если количество вершин n делится нацело на количество транспортных средств k , то количество вершин для каждого транспортного средства будет одинаковым и равно отношению n/k .

В противном случае k_1 маршрутов должны содержать $\lfloor n/k + 1 \rfloor$ вершин, где k_1 — остаток от деления n на k , и $k - k_1$ транспортных средств должны посетить ровно $\lfloor n/k \rfloor$ вершин-клиентов.

При допущении условия, что все вершины-клиенты имеют одинаковую потребность в товаре, появляется возможность неявно управлять максимальной грузоподъемностью транспортных средств, поскольку количество маршрутов k может быть заданным заранее. Если в ходе планирования развозки выясняется, что маршруты будут содержать слишком большое количество вершин, требующее очень большой грузоподъемности транспортного средства для доставки товара, тогда количество маршрутов необходимо увеличить. На практике увеличение количества маршрутов не предполагает обязательного увеличения количества транспортных средств. Два маршрута могут рассматриваться как путь одной машины с дополнительным возвращением в депо для загрузки новой партией товара.

Обратим также внимание, что существует ещё один случай, когда число k может содержать не точное количество транспортных средств. Это связано с тем, что предлагаемый ниже алгоритм на основе дихотомической процедуры деления вершин на группы является только алгоритмом кластеризации, т. е. строит разбиение вершин, но для получения конкретных маршрутов требуется последующее решение ЗК. После проведения кластеризации для некоторого заданного k возможно объединение нескольких из полученных групп на основе некоторого критерия близости, что позволит обрабатывать ситуации, когда вместимость части транспортных средств кратно превосходит другие, а ограничение на время объезда маршрута несущественно.

Поскольку такие ситуации встречаются на практике относительно редко, на них подробно не останавливаемся.

2.3 Общий вид процедуры дихотомического деления вершин на группы

Рассмотрим общий вид процедуры дихотомического деления вершин на группы для каждого транспортного средства. Она послужит некоторой алгоритмической схемой, на основе которой, как показал вычислительный эксперимент, удалось предложить новые алгоритмы для четырёх видов ЗМТ, перечисленных в начале этой главы.

Процедура предназначена для двухфазных методов решения ЗМТ, в которых фаза кластеризации предшествует построению маршрутов (см. разд. 1.3). Выбор метода, используемый для решения ЗК на второй фазе, здесь не рассматривается. Это может быть любой из известных алгоритмов, выбранный на основе требуемого соотношения качества и времени работы.

В своих основных чертах процедура была исследована на основе СЗМТ, и в дальнейшем для разработки алгоритмов, таких как, например, алгоритм для решения ЗМТУГ, использовался уже наилучший вариант, что позволило значительно уменьшить объём необходимой работы. Именно в таком виде она приводится в этом разделе, список некоторых других неудачных направлений описан в разд. 2.5.

На вход процедуры поступает множество V вершин для деления на заданное количество групп k , а также симметричная матрица стоимости переездов C , содержащая неотрицательные элементы. На её главной диагонали могут быть только нули. Процедура является рекурсивной и на каждом шаге производит деление вершин на две группы.

Основные этапы следующие:

1. Выполняем поиск двух вершин, стоимость переезда между которыми максимальна среди всех возможных пар. Найденные вершины послужат начальным наполнением будущих групп.
2. Все оставшиеся вершины должны быть распределены между двумя группами. Для этого по очереди производится поиск ближайшей вершины сначала к первой группе, затем ко второй и т. д. Понятие близости вершины к группе может быть определено несколькими способами, о которых пойдёт речь ниже. Если количество вершин или груза, с ними связанного, в группах должно быть неравномерным, то необходимо произвести предварительное начальное наполнение одной из групп для выравнива-

ния на основе выбранного критерия близости.

3. Если одна или обе полученные группы содержат вершины более, чем для одного транспортного средства, то необходимо повторить работу процедуры при помощи рекурсивного вызова.

2.4 Процедура дихотомической кластеризации для СЗМТ

Рассмотрим алгоритм, способный находить приближённое решение СЗМТ. Этот алгоритм не имеет родственников среди тех, которые предназначены для решения несбалансированных видов ЗМТ. Ключевую роль здесь играет возможность предварительного расчёта количества вершин для маршрута каждого транспортного средства. Как это принято во всех двухфазных алгоритмах с кластеризацией на первом этапе, в этом методе вначале все множество n вершин-клиентов делится на k групп, а затем для каждой группы с добавлением депо вычисляется k замкнутых маршрутов путём решения ЗК любым подходящим методом.

Первым шагом в процессе работы необходимо определить количество вершин для каждого будущего маршрута. Эта задача обсуждалась в разд. 2.2. Примем, что количество вершин для каждого транспортного средства задаётся массивом $M = m_1, m_2, \dots, m_k$. Для СЗМТ этот массив может содержать только числа, не отличающиеся более, чем на единицу.

Процесс кластеризации можно представить в виде рекурсивной процедуры, которая получает на входе множество $V = \{v_1, v_2, \dots, v_n\}$ из n целевых вершин-клиентов, желаемое число групп k и массив M из k элементов, определяющий количество вершин в каждой группе. Предполагается возможность определения стоимости проезда между любой парой целевых вершин.

Процедура при каждом вызове производит деление на две группы и выполняет рекурсивный спуск при дальнейшей необходимости. Число k не должно быть меньше двух, в противном случае задача кластеризации не имеет смысла. Элементы массива M должны быть строго больше нуля.

1. Определим количества вершин для двух будущих групп, обозначим эти числа как l_1 и l_2 :
 - (a) если $k = 2$, то установим $l_1 = m_1$, $l_2 = m_2$ и остановим процесс вычисления этих параметров. В дальнейшей работе алгоритма тре-

буется, чтобы $l_1 \geq l_2$, поэтому если это не так, выполним обмен их значений;

(b) установим $l_1 = 0$ и l_2 равным сумме всех элементов массива M ;

(c) выполним цикл при q равным от 1 до k , на каждом шаге которого к l_1 будем прибавлять значение m_q , а из l_2 — это же значение вычитать. Остановим работу цикла, если $l_1 \geq l_2$. Последнее значение счётчика цикла q необходимо также запомнить, т. к. оно понадобится в будущем для принятия решения о дальнейших рекурсивных вызовах.

2. Среди вершин множества V выберем две v_i и v_j с максимальной стоимостью проезда от одной к другой среди всех возможных пар вершин множества V . Выбранные вершины послужат начальным наполнением двух будущих групп. Вершину v_i поместим в первую группу, а v_j — во вторую. Из множества V вершины v_i и v_j должны быть исключены.
3. Введём понятие близости некоторой вершины v из множества V до каждой из двух строящихся групп G_1 и G_2 : пусть g_1 — вершина с наименьшей стоимостью переезда из v среди всех вершин из G_1 , и, аналогично, g_2 — вершина с наименьшей стоимостью переезда из v среди всех вершин из G_2 . Близостью вершины v к группе G_1 является стоимость переезда из v в g_1 с вычитанием стоимости переезда из v в g_2 . Таким же образом можно определить близость вершины v к группе G_2 как стоимость переезда из v в g_2 с вычитанием стоимости переезда из v в g_1 . Обратим внимание, что эта величина может быть отрицательной и это считается допустимым.
4. Производим построение вспомогательного массива структур D , содержащего столько элементов, сколько вершин включает множество V . Этот массив должен хранить значения близости каждой вершины из V до каждой из двух групп. Произвести такое построение на текущем этапе можно за линейное время, т. к. две группы пока содержат только по одному элементу.
5. $l_1 - l_2$ раз повторим следующее действие: среди вершин множества V выбирается такая, которая имеет наименьшее значение близости до первой группы, затем эта вершина добавляется в первую группу, удаляется из V и удаляется также соответствующий элемент из массива D . После каждой операции добавления необходимо произвести обновление массива D ,

если вновь добавленная вершина стала ближайшей при расчёте близости для некоторого элемента D , то необходимо для него выполнить пересчёт. Производить поиск вершины с наименьшим значением близости нужно на основе информации из D .

6. $l_2 - 1$ раз повторяются следующие действия:

- (a) на основе данных из D выбирается вершина, принадлежащая V , с наименьшим значением близости к первой группе. Затем эта вершина добавляется к первой группе, удаляется из V , удаляется соответствующий элемент из D и производится обновление элементов D , как это было описано выше;
- (b) на основе данных из D выбирается вершина, принадлежащая V , с наименьшим значением близости ко второй группе. Затем эта вершина добавляется к второй группе, удаляется из V , удаляется соответствующий элемент из D и производится обновление элементов D .

7. Предполагается, что первая группа содержит вершины для q транспортных средств, а вторая группа — для $k - q$ транспортных средств. Если одно из этих значений больше единицы, то необходимо продолжить работу путём выполнения рекурсивного вызова. При этом для первой группы множество V при рекурсивном вызове должно содержать элементы первой группы, $k = q$, а массив m содержать элементы массива m с первого до q -того. Для второй группы множество V должно содержать элементы второй группы, $k = k - q$, а массив m должен содержать элементы с $q + 1$ до k .

После того, как все рекурсивные вызовы выполнены и получены k групп вершин, в каждую из них необходимо добавить вершину-депо.

Трудоёмкость метода при разделении вершин на k групп в обоих случаях имеет порядок $O(n^2)$, она выводится из рекуррентного соотношения $T(n) = 2T(n/2) + n^2$.

Напомним, что это только процедура кластеризации, необходимо произвести построение циклических маршрутов путём решения ЗК. Достаточно хорошие результаты получаются при применении приближённого алгоритма Лина-Кернигана [68]. Пример работы такого сочетания алгоритмов приведён на рис. 6.

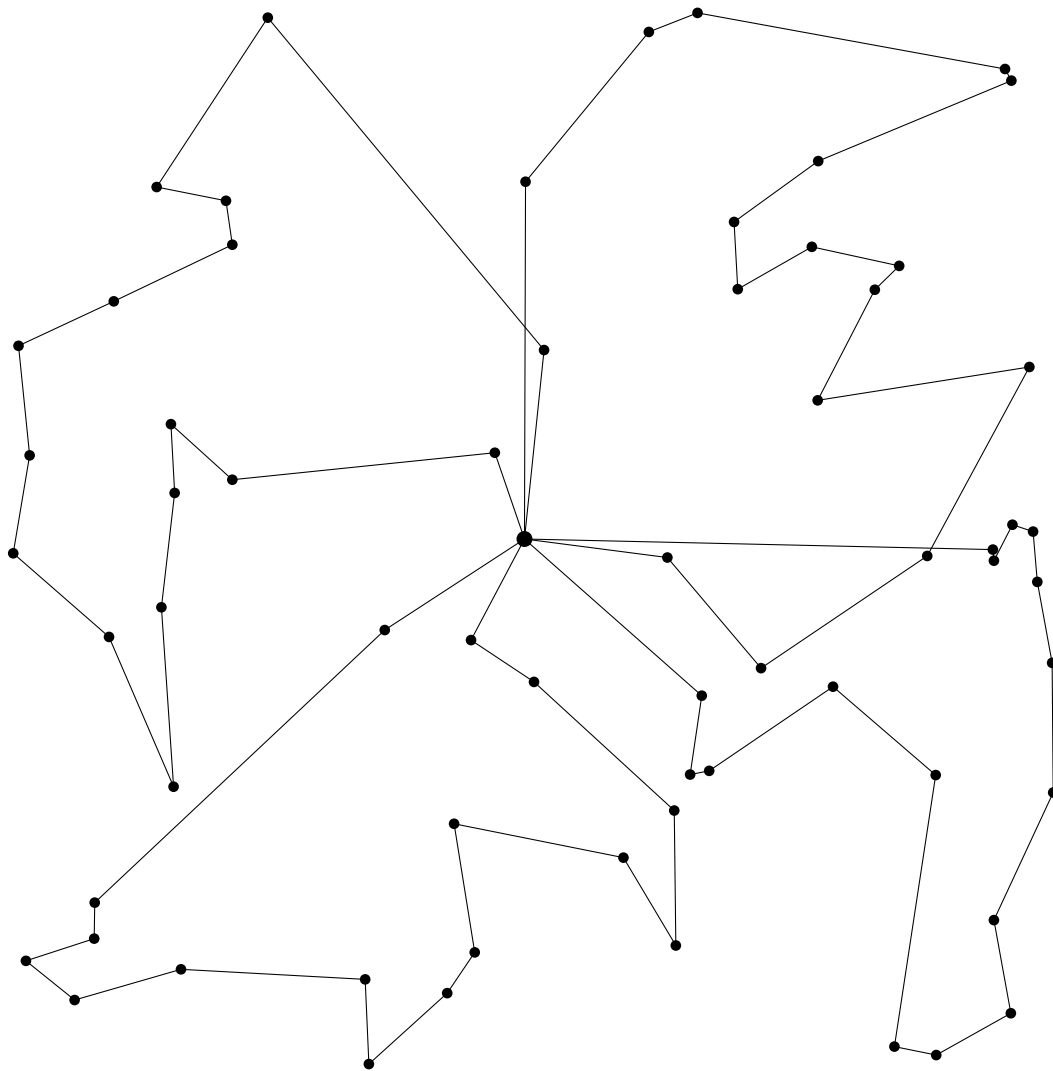


Рис. 6. Пример построения четырёх маршрутов для посещения 64 вершин алгоритмом дихотомической кластеризации с последующим применением алгоритма Лина-Кернигана.

2.5 Другие варианты дихотомического алгоритма для СЗМТ

В предыдущем разделе приведён наиболее удачный вариант алгоритма дихотомической кластеризации, но был исследован ещё ряд подходов, показавших ухудшение результатов и приведших к выбору именно описанного варианта. Развитие этого направления дало возможность получить развитый алгоритм с учётом грузоподъёмности, описанный в разд. 2.10. Приведём вкратце основные направления поиска, не показавшие хороших результатов:

1. Попытка определить близость вершины v к некоторой группе G равной стоимости переезда от v до g/inG , где g — это вершина G с наименьшей стоимостью переезда до v , приводит к заметному ухудшению результата.
2. Смена дихотомического деления на трихотомическое деление, в котором на каждом шаге производится построение трёх групп также приводит к ухудшению результата. Можно предположить, что дальнейшее увеличение количество групп для деления будет также ухудшать результат. Кроме этого, трихотомическое деление обладает трудоёмкостью каждого шага $O(n^3)$, деление на четыре группы — $O(n^4)$ и т. д.
3. Учёт при дихотомическом делении положения депо и обработка стоимости переезда до него при определении близости вершины к группе — ещё одно направление, приводящее к ухудшению результата.

В исследовании алгоритмов для решения ЗМТУГ и ЗМТ для нескольких депо никаких попыток рассмотрения других вариантов дихотомической процедуры не предпринималось. За основу был выбран вид, описанный в предыдущем разделе.

2.6 Обменная оптимизация при дихотомическом делении

После проведения деления вершин на две группы на некотором рекурсивном вызове процедуры, описанной в разд. 2.4, возможно выполнение дополнительных оптимизаций, улучшающих качество кластеризации.

Для примера опишем так называемую обменную оптимизацию.

1. Рассмотрим пару вершин: вершину i из первой группы и вершину j из второй группы.
2. Выполним пробный обмен этих вершин между группами, т. е. поместим вершину i в вторую группу, а j — в первую.

3. Если такой обмен приводит к улучшению маршрутов, то оставляем вершины в новом варианте, в противном случае восстанавливаем их исходное положение.

Здесь необходимо решить вопрос, как оценивать изменения длины маршрутов. Можно использовать два варианта:

1. Выполнять пробное решение ЗК каким-либо быстрым алгоритмом, например 2-опт [67].
2. Оценивать возможную длину маршрута путём использования оценки снизу, которая может быть вычислена как сумма длин рёбер минимального остова, построенного на полном графе из всех вершин, с добавлением длины самого длинного ребра.

Такую обменную оптимизацию имеет смысл применять после всех шагов деления вершин на две группы. Оценим трудоёмкость этой процедуры. Пусть в первой группе n_1 вершин, а во второй группе — n_2 . Тогда попытку обмена придется выполнить $n_1 n_2$ раз. Если трудоёмкость каждой попытки имеет порядок $O(n_1^2 + n_2^2)$, то общая трудоёмкость при всех делениях на k групп будет порядка $O(n^4)$. Для уменьшения трудоёмкости можно, практически не ухудшая результат, как наблюдалось при тестировании, проверять не все вершины, попавшие в первую и вторую группы, а только часть из них (например, $1/\sqrt{n}$ часть). При этом имеет смысл выбирать именно те вершины, которые были отнесены к соответствующим группам в самую последнюю очередь. При этом общая трудоёмкость будет порядка $O(n^3)$.

2.7 Алгоритм разрезания общего маршрута для СЗМТ

Рассмотрим две модификации известных алгоритмов для СЗМТ. Алгоритм разрезания общего маршрута и алгоритм заметания — это два хорошо известных алгоритма, ранее разработанных для стандартного вида ЗМТ. Выбранные алгоритмы являются относительно несложными и хорошо подходят для рассмотрения, а также могут быть добавлены в вычислительный эксперимент для сравнения качества алгоритма дихотомической кластеризации. Адаптация развитых алгоритмов, таких как, например, алгоритм лепестков (см. разд. 1.5.4) потребует уже значительных изменений и в настоящей работе не рассматривается.

Возможность решения ЗМТ путём разрезания общего маршрута, полученного при построении общего пути коммивояжёра на множестве всех вершин, была описана в разд. 1.5.5, Приведём вариант этого алгоритма для СЗМТ.

Алгоритм содержит следующие этапы:

1. Вычисление общего маршрута коммивояжера по n вершинам (исключая депо). Алгоритм для выполнения этой операции можно выбрать среди известных алгоритмов решения ЗК на основе времени их работы и качества получаемых решений. Таким алгоритмом может быть, например, алгоритм дерева с трудоёмкостью $O(n^2)$ и точностью 2 [61], алгоритм Кристофидеса с трудоёмкостью $O(n^3)$ и точностью 1,5 [3], алгоритм Лина-Кернигана.
2. Вычисление количества вершин для каждого транспортного средства (см. разд. 2.2). Результат расчёта должен быть представлен в виде массива $M = m_1, m_2, m_3, \dots, m_k$.
3. Разрезание общего маршрута на k фрагментов в соответствии с результатом, полученном на предыдущем шаге, с минимизацией суммарной стоимости переездов, соответствующих фрагментам маршрута. Пусть нумерация вершин в маршруте: g_1, g_2, \dots, g_n . Обозначим звенья в маршруте: $(g_1, g_2), (g_2, g_3), \dots, (g_{n-1}, g_n)$. Из них выбирается группа звеньев $(g_i, g_{i+1}), (g_{i+m_1}, g_{i+m_1+1}), (g_{i+m_1+m_2}, g_{i+m_1+m_2+1}), \dots$ с максимальной суммарной стоимостью переезда при изменении i от 1 до $\max M$. При этом стоимость переезда, соответствующей оставшимся фрагментам маршрута будет минимальной.

Возможен иной способ оценки длины совокупности маршрутов путём вычисления суммарного веса минимальных остовов, построенных на вершинах, вошедших в выделенные фрагменты, но обладающий значительно большей трудоёмкостью.

4. Вычисление k замкнутых маршрутов по депо и вершинам, вошедшим в каждый из фрагментов общего маршрута. Первый из них строится на множестве вершин, в которое входит депо и вершины $g_{i+1}, g_{i+2}, \dots, g_{i+m_1}$, второй маршрут — на множестве вершин, в которое входит депо и вершины $g_{i+m_1+1}, g_{i+m_1+2}, \dots, g_{i+m_1+m_2}$ и т. д. Для этих маршрутов также можно использовать подходящий приближенный алгоритм.

Трудоёмкость алгоритма определяется в основном трудоёмкостью первого этапа, т. к. третий этап можно выполнить за время $O(n)$, а трудоёмкость четвёртого не менее, чем в k раз меньше трудоёмкости первого, если на всех этапах используется один и тот же приближенный алгоритм решения ЗК. Время работы второго этапа всегда $O(k)$. Обратим внимание, что возможность решения третьего этапа за время $O(n)$ является следствием из сбалансированной природы задачи. Как говорилось в разд. 1.5.5 в несбалансированном виде этот этап требует времени $O(n^2)$.

Существует ещё один способ реализации третьего и четвёртого этапов алгоритма: для каждого из $\max M$ вариантов разрезания общего маршрута сразу вычисляем k маршрутов коммивояжера и находим их общую длину. В этом случае, опробовав все $\max M$ вариантов разрезания, среди них можно выбрать наилучший. Однако при этом трудоёмкость алгоритма возрастет на порядок.

На рис. 7 приведён пример работы алгоритма сбалансированного разрезания общего маршрута, построенного алгоритмом Лина-Кернигана.

2.8 Алгоритм заметания для СЗМТ

В этом разделе приводится особый алгоритм, не представляющий из себя особой ценности для практического применения, но достаточно полезный, для иллюстрации влияния сбалансированного подхода. Особенность этого метода [8] заключается в том, что он предназначен для модифицированной ЗМТ. В работе этого алгоритма предполагается доступность информации о расположении всех вершин на плоскости. Здесь ключевым свойством является наличие информации об углах между вершинами относительно депо. Очевидно, что задача в таком варианте подходит не для всех практических применений, т. к. обычно используется граф транспортной сети, а не вершины на плоскости. Тем не менее, такой вариант широко обсуждается и известны развитые алгоритмы на его основе (см. разд. 1.5.4), дающие качественные результаты.

Рассмотрим алгоритм деления n вершин на k групп по m вершин в каждой для случая, когда вершины являются точками на плоскости XOY , а расстояния между ними — евклидовы:

1. Пусть депо находится в центре координат: $((x_\sigma = 0, y_\sigma = 0))$, а координаты

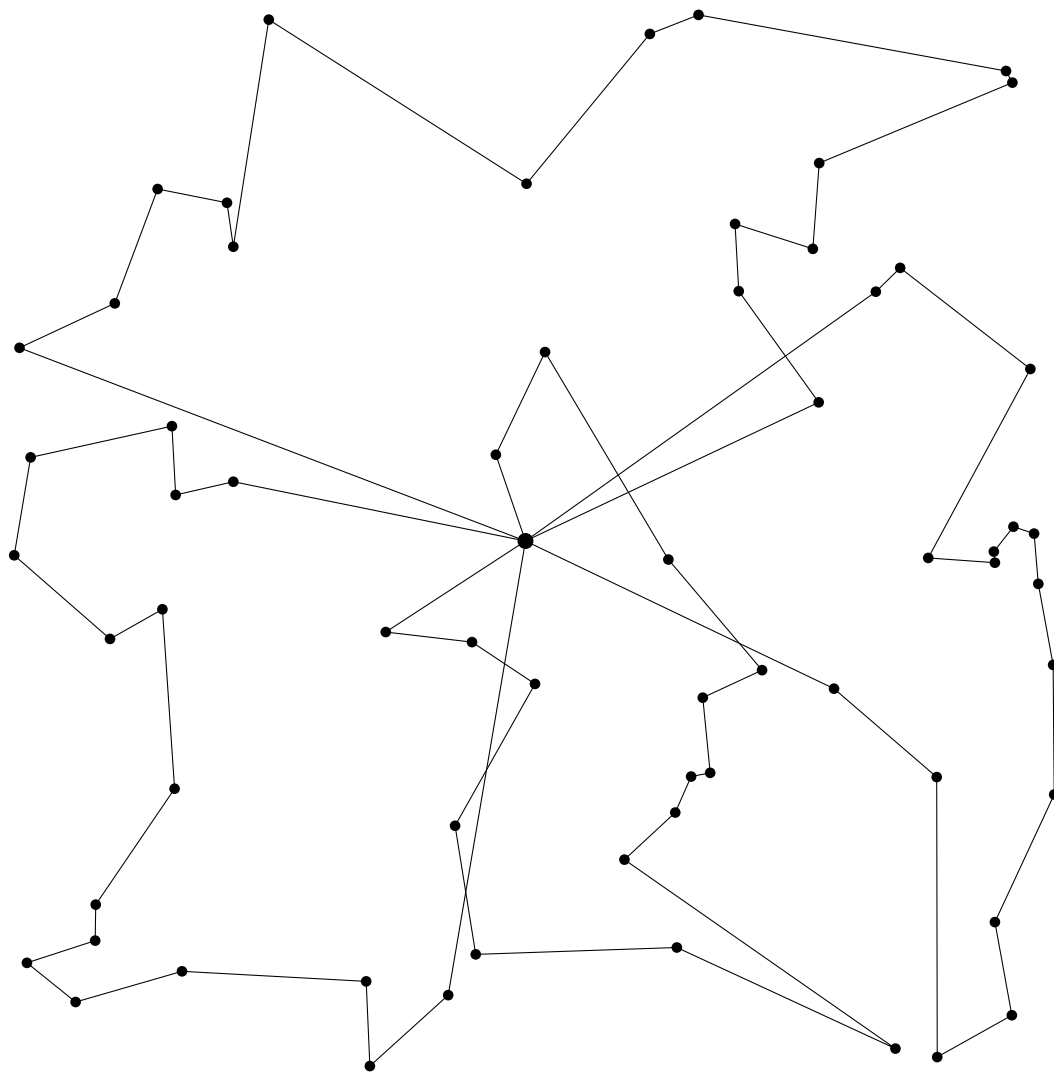


Рис. 7. Пример построения четырёх маршрутов для посещения 64 вершин алгоритмом разрезания общего маршрута, построенного алгоритмом Лина-Кернигана.

всех остальных вершин (x_i, y_i) одновременно не равны нулю.

2. Перейдем для n вершин к полярной системе координат (r_i, α_i) и упорядочим все вершины (кроме депо) по углу $\{\alpha_i\}$, $0 \leq \alpha_i \leq 2\pi$, причем вершины с одинаковым углом упорядочим по радиусу $\{r_i\}$. В результате получим некоторый маршрут коммивояжера по n вершинам.
3. Вычисляем количество вершин для каждого транспортного средства (см. разд. 2.2), и записываем результат в массив $M = m_1, m_2, \dots, m_k$.
4. Разрезаем полученную последовательность вершин на k фрагментов по m_i вершин в каждом фрагменте, где $1 \leq i \leq k$, с минимизацией суммарной длины частей маршрута. Вычисление варианта разрезания с наименьшей длиной можно произвести путём циклического сдвига, как это описывалось в разд. 2.7.
5. Находим k замкнутых маршрутов по депо и вершинам, вошедшим в каждый из фрагментов маршрута, т. е. по вершинам, попавшим в сектор (по углу).

В этом алгоритме трудоемкость определяется вычислением k замкнутых маршрутов на последнем этапе, т. к. трудоемкость перехода к полярной системе координат $O(n)$, трудоемкость упорядочения — $O(n \cdot \log n)$, трудоемкость разрезания — $O(n)$. Последний этап также можно выполнить с трудоемкостью $O(n \cdot \log n)$, если перед построением очередного маршрута по группе вершин вычислить триангуляцию по точкам (вершинам, вошедшим в группу, включая депо), затем по триангуляции построить минимальный остов, а по остову — маршрут алгоритмом дерева [61].

Пример работы описанного алгоритма приведён на рис. 8.

2.9 Вычислительный эксперимент для СЗМТ

Для взаимного сравнения некоторых из предложенных алгоритмов был проведён вычислительный эксперимент со следующими характеристиками:

1. n вершин генерировались, как точки на плоскости XOY в единичном квадрате с независимыми равномерно распределёнными координатами.
2. Депо помещалось в центр квадрата.

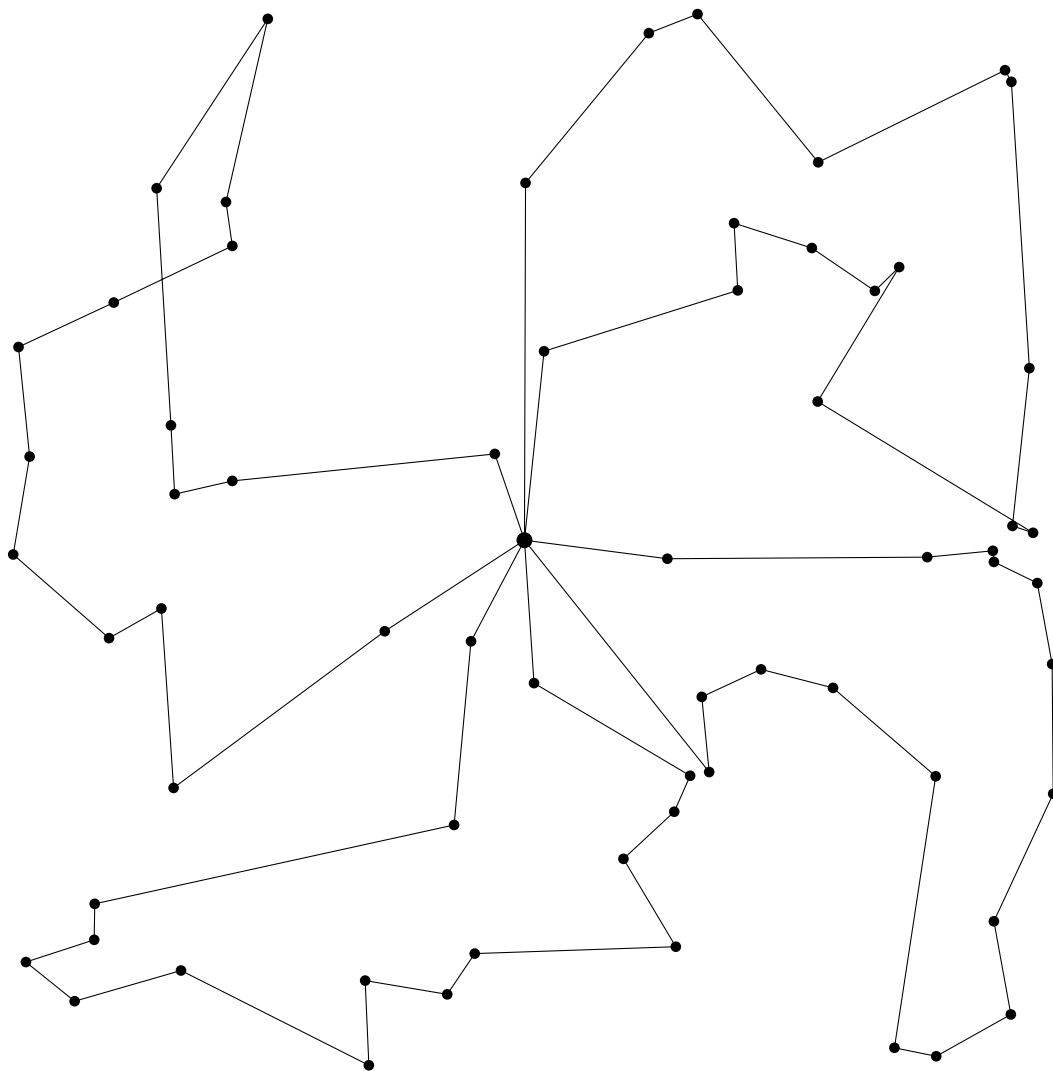


Рис. 8. Пример построения четырёх маршрутов для посещения 64 вершин сбалансированным алгоритмом заметания.

3. Оценка качества полученного решения вычислялась как отношение общей длины всех k маршрутов к нижней оценке общего маршрута по всем вершинам, включая депо.
4. Количество реализаций подбиралось таким образом, чтобы среднеквадратичное отклонение оценки не превышало нескольких единиц в последней значащей цифре оценки.

В качестве нижней оценки использовалась сумма длин ребер минимального остова, в котором самое длинное ребро учтено дважды. Для вычисления ЗК на промежуточных шагах алгоритмов использовалась одна и та же последовательность:

1. Алгоритм дерева [61], строящий маршрут по минимальному остову.
2. Пятикратное повторение алгоритма 2-опт на полученном маршруте [67].
3. Локальная оптимизация с окном из девяти вершин [61].

В таблицах приводятся результаты для следующих алгоритмов:

1. Алгоритм разрезания общего маршрута (табл. 1).
2. Алгоритм дихотомической кластеризации с выбором по минимальной стоимости переезда (табл. 2).
3. Алгоритм дихотомической кластеризации с выбором по разности минимальной стоимости переездов (табл. 3).
4. Сбалансированный алгоритм заметания (табл. 4).

Результаты вычислительного эксперимента позволяют сделать следующие выводы:

1. По качеству решений алгоритмы расположились в следующем порядке (от худшего к лучшему): “сбалансированное разрезание общего маршрута”, “дихотомическая кластеризация по минимуму”, “дихотомическая кластеризация по разности”. Сбалансированный алгоритм заметания превзошел все остальные при $k = 16$ и менее, но уступил другим при больших k .
2. При решении задачи с использованием матрицы стоимости переездов наиболее перспективным представляется использование алгоритма “кластеризация по разности”, который можно реализовать с трудоемкостью $O(n^2)$.

Таблица 1. Результаты вычислительного эксперимента для сбалансированного алгоритма разрезания общего маршрута. Приводится среднее значение качества решений по всем выборкам. Достоверность значений равна 0,95 с двумя значащими цифрами после запятой.

Вершин / экипажей	2	4	8	16	32	64
32	1,59	1,98	-	-	-	-
64	1,51	1,80	2,35	-	-	-
128	1,43	1,63	2,01	2,78	-	-
256	1,38	1,55	1,84	2,37	3,49	-
512	1,33	1,44	1,64	2,02	2,80	4,42
1024	1,32	1,38	1,51	1,79	2,34	3,46

Таблица 2. Результаты вычислительного эксперимента для алгоритма дихотомической кластеризации по минимуму. Приводится среднее значение качества решений по всем выборкам. Достоверность значений равна 0,95 с двумя значащими цифрами после запятой.

Вершин / экипажей	2	4	8	16	32	64
32	1,45	1,82	2,19	-	-	-
64	1,40	1,67	2,19	-	-	-
128	1,35	1,55	1,93	2,70	-	-
256	1,31	1,47	1,77	2,34	3,52	-
512	1,29	1,43	1,67	2,08	2,91	4,55
1024	1,30	1,39	1,58	1,91	2,52	3,68

Таблица 3. Результаты вычислительного эксперимента для алгоритма дихотомической кластеризации по разности. Приводится среднее значение качества решений по всем выборкам. Достоверность значений равна 0,95 с двумя значащими цифрами после запятой.

Вершин / экипажей	2	4	8	16	32	64
32	1,31	1,60	-	-	-	-
64	1,29	1,47	1,94	-	-	-
128	1,28	1,38	1,70	2,44	-	-
256	1,26	1,33	1,55	2,06	3,14	-
512	1,25	1,29	1,44	1,79	2,54	4,11
1024	1,25	1,27	1,37	1,62	2,13	3,22

Таблица 4. Результаты вычислительного эксперимента для сбалансированного алгоритма заметания. Приводится среднее значение качества решений по всем выборкам. Достоверность значений равна 0,95 с двумя значащими цифрами после запятой.

Вершин / экипажей	2	4	8	16	32	64
32	1,31	1,56	-	-	-	-
64	1,26	1,41	1,92	-	-	-
128	1,28	1,34	1,65	2,50	-	-
256	1,27	1,29	1,48	2,00	3,38	-
512	1,25	1,28	1,37	1,66	2,59	4,68
1024	1,24	1,26	1,32	1,48	2,02	3,51

2.10 Сбалансированный алгоритм кластеризации для ЗМТУГ

Перейдём к рассмотрению сбалансированного алгоритма, предназначенного для решения ЗМТУГ в её общем виде, но имеющего тенденцию к сбалансированию загрузки транспортных средств. Эта задача является более сложной, чем поиск алгоритмов для СЗМТ, т. к. для стандартной ЗМТ уже предложено большое количество различных подходов, показывающих очень хорошее качество.

При исследовании методов необходимо постоянно помнить, что качество алгоритмов не является единственным критерием для оценки. Многие из популярных алгоритмов, дающих удачные решения, требуют больших затрат памяти и процессорного времени [43].

Тем не менее, трудности не ограничиваются только расходами вычислительных ресурсов. При реализации метаэвристик часто появляется проблема подбора управляющих параметров. Они имеют сильное влияние на качество, а найти универсальное их сочетание практически не удаётся. Авторы соответствующих идей на основе метаэвристик предлагают различные эмпирические способы расчёта значений, с наибольшей вероятностью дающих хороший результат в зависимости от параметров входных данных, но и эти оценки нередко приводят к ложным выводам. Часть алгоритмов имеет один-два дискретных параметра, допускающих перебор их значений и выбор самого лучшего результата среди всех вариантов.

Сложившаяся ситуация даёт очень большой простор для исследований. Алгоритмы должны не только находить качественные результаты, но и быть пригодными к внедрению в программные пакеты, ориентированные на самый широкий круг пользователей. Все методы, которые представляют из себя не точные алгоритмы, а только алгоритмические схемы, какими являются метаэвристические стратегии, могут использоваться только опытными операторами, которые имеют детальное представление о природе алгоритма и способны произвести необходимый подбор параметров для получения качественного решения.

При современном уровне развития техники трудности существуют даже для небольших задач (менее 200 вершин), т. к. процедура вариации параметров может быть достаточно длительной и возможно только в случае, если параметры имеют дискретное множество значений. Для больших задач (до 1000 вершин и более) проблема становится ещё более острой. Произво-

дить глубокий поиск или вести пробный запуск разных алгоритмов нельзя — время работы растёт с неприемлемой скоростью.

В приводимом ниже алгоритме пространство поиска ограничено условием сбалансирования. Алгоритм не содержит никаких параметров и может решать большие задачи за относительно небольшое время. Появляется вопрос, каким образом повлияет применение дополнительных ограничивающих условий на качество результатов. Как покажет вычислительный эксперимент, новый алгоритм проигрывает в качестве развитым метаэвристикам в области небольших задач, но при росте размерности результаты начинают улучшаться при сохранении экономии времени работы.

Предлагаемый алгоритм содержит две фазы: в начале решается задача кластеризации (разделение общего множества вершин на группы для каждого транспортного средства), затем выполняется построение конечных маршрутов путём решения традиционной ЗК. Таким образом, его также можно отнести к классу кластерных алгоритмов с процедурой кластеризации, предшествующей построению маршрутов (см. разд. 1.3). Количество транспортных средств заранее не определяется, а вычисляется в ходе работы на основе информации о их грузоподъёмности. Применяемый способ решения ЗК может быть выбран произвольно среди известных методов.

Ниже приведено подробное описание сбалансированной процедуры кластеризации. Основная сложность в реализации алгоритма связана с тем, что в его работе постоянно учитываются две величины: стоимость маршрута и суммарная потребность в товаре вершин-клиентов, включенных в маршрут. Стремление к наилучшему распределению загрузки транспортных средств может негативно сказываться на стоимости маршрута, и наоборот. Другая сложность заключается в том, что потребность в товаре у клиентов может быть достаточно неравномерной, и существует риск, что добавление вершин на основе оценки стоимости решения в самый последний момент нарушит ограничение грузоподъёмности.

Рассмотрим описание алгоритма. Дано множество из n вершин $V = \{v_1, \dots, v_n\}$. Для каждой вершины v_i , принадлежащей V , задана величина потребности в товаре $Q(v_i) > 0$. Все транспортные средства имеют заданную заранее одинаковую грузоподъёмность q . Также задаётся особая вершина — депо v_0 , представляющая точку начала и конца всех построенных маршрутов. Для общности полагаем, что $Q(v_0) = 0$. Для любой пары вершин i и

j из $\{v_0, v_1, \dots, v_n\}$ определены расстояния (стоимость переезда) $D(i, j)$ из i в j . Будем считать, что расстояния метрические, т. е. они симметричны и для них выполняется условие треугольника.

Алгоритм строит множество $R = \{r_1, \dots, r_k\}$ из k замкнутых маршрутов (k не задано) таких, что:

1. Вершина v_0 принадлежит каждому из k маршрутов.
2. Каждая из вершин v_i , принадлежащая множеству V , принадлежит одному и только одному из k маршрутов;
3. Для каждого из k маршрутов r_i , которому принадлежат $m(i)$ вершин $\{v_0, v_{i_1}, v_{i_2}, \dots, v_{i_{m(i)}}\}$, должно выполняться следующее условие:

$$\sum_{j=1}^{m(i)} Q(v_{i_j}) \leq q. \quad (4)$$

Еще раз заметим, что первая фаза алгоритма является всего лишь алгоритмом кластеризации, и для получения окончательных маршрутов необходимо решить ЗК для каждого из k множеств r_i .

В идеале алгоритм должен получать такое множество маршрутов R , чтобы их суммарная длина была минимальной. Обычно такое требование недостижимо, поэтому алгоритм должен хотя бы приближаться к идеалу. Так, чаще всего, если удастся разделить все множество вершин на меньшее число маршрутов, то и их суммарная длина будет меньше. Кластеризацию будем производить следующим образом: вначале разделим все вершины множества V на две группы, затем каждую из групп снова на две группы и т. д. до тех пор, пока каждая из групп не будет удовлетворять условию (4).

2.11 Рекурсивная процедура дихотомического сбалансированного разделения вершин

В ходе вычислений используется рекурсивная процедура $divide(V', k')$, выполняющая разделение вершин. Для соблюдения однозначности параметры процедуры обозначаем буквами с апострофами. Процедура должна выполнить разделение множества вершин V' на k' групп. При использовании процедуры множество V' будет некоторым подмножеством из V . На каждом

этапе производится деление только на две группы, а затем выполняется рекурсивный вызов с целью продолжить разделение. Процедуру $divide(V', k')$ можно представить в следующем виде:

1. Инициализация некоторых параметров. Переменная q_t должна содержать величину общей потребности в товаре для всех вершин из V' . Переменные k'_1 и k'_2 хранят значение, вершины для какого количества экипажей должны быть отнесены к первой и второй группам, они вычисляются по формулам:

$$k'_2 = \lfloor \frac{k'}{2} \rfloor,$$

$$k'_1 = k' - k'_2.$$

Переменные Δ_1 и δ_1 обозначают верхнюю и нижнюю границы суммы количества товара тех вершин, которые могут быть отнесены к первой группе при разделении, аналогично справедливо и для Δ_2 и δ_2 для второй группы. Их значения вычисляются следующим образом:

$$\Delta_1 = k'_1 \cdot q,$$

$$\Delta_2 = k'_2 \cdot q,$$

$$\delta_1 = q_t - \Delta_2,$$

$$\delta_2 = q_t - \Delta_1.$$

Обратим внимание, что значения δ_1 или δ_2 могут оказаться отрицательными. В этом случае величина k' уменьшается на 1 и вычисления повторяются.

2. Далее ищутся две вершины s_1 и s_2 с максимальной стоимостью переезда между ними. Вершины s_1 и s_2 являются начальным наполнением будущих групп. Если k'_1 не равно k'_2 , то приводимое ниже описание алгоритма необходимо выполнить два раза с обменом значений s_1 и s_2 и выбором наилучшего варианта. Вопрос, как сравнить качество двух вариантов, обсуждается ниже.
3. Сначала рассмотрим разбиение в общем случае, исключая $k' = 2$ и $k' = 3$, разбиение для последних описано отдельно. Итак, вершины s_1 и s_2 являются начальным наполнением будущих групп. Далее на основе различных критериев мы будем относить оставшиеся вершины из V' в первую

или вторую группу. Важную роль играет понятие близости вершины к группе. Используется оценка, равная разности расстояний между некоторой вершиной до вершины, ближайшей из первой группы, и вершиной, ближайшей из второй группы. Эта оценка аналогична той, которая была определена в разд. 2.3. При практической реализации вычисления такой оценки для каждой вершины, еще не отнесенной к какой-либо группе, должны храниться ссылки на ближайшие вершины из обеих групп, и после перемещения вершины в группу эти ссылки должны обновляться.

4. Если $k'_1 \neq k'_2$, то выполняем добавление к первой группе ближайших к ней вершин до тех пор, пока не будет достигнута величина средней загрузки экипажа, которая вычисляется как q_t/k' .
5. В дальнейших вычислениях потребуются значения желаемого количества товара для каждой группы. Получить эти значения можно путём умножения чисел k'_1 и k'_2 на величину средней загрузки транспортного средства. До тех пор, пока количество товара у вершин, отнесенных к первой группе, меньше желаемого количества для первой группы, и количество товара вершин, отнесенных к второй группе, меньше желаемого уровня для второй группы, производим следующие действия:
 - (а) если запас для первой группы, равный разнице желаемого уровня её загрузки и суммы текущего наполнения, больше запаса для второй группы, вычисленного аналогичным способом, и добавление к первой группе ближайшей к ней вершины не приведёт к превышению желаемого уровня, то относим к первой группе ближайшую к ней вершину;
 - (б) если добавление ко второй группе ближайшей к ней вершины не приведёт к превышению желаемого уровня наполнения, то относим ко второй группе ближайшую к ней вершину;
 - (в) вычислим два значения для каждой группы, равных оставшемуся резерву в каждой из них после добавления к ней текущей ближайшей вершины. Необходимо обратить внимание, что в общем случае каждое из значений может оказаться отрицательным, поэтому обязательно используем их модули. Если полученное значение для первой группы меньше, чем для второй, то относим к первой группе ближайшую к ней вершину, в противном случае — ко второй.

6. Пока остаются нераспределённые вершины, относим их к той группе, где не произошло превышения желаемого уровня загрузки.
7. Полученные уровни загрузки групп должны быть обязательно не меньше, чем δ и не больше, чем Δ , которые были вычислены выше (Δ_1, δ_1 для первой группы и Δ_2, δ_2 — для второй). Если какая либо группа нарушает это условие, то необходимо произвести процедуру балансировки:
 - (a) просматриваем вершины той группы, где произошло превышение Δ . Обратим внимание, что превышение этого значения у обеих групп одновременно невозможно. Порядок просмотра должен быть обратным тому, как происходило приписывание вершин группе;
 - (b) если перенос некоторой вершины в противоположную группу не приводит к превышению значения в ней, то такой перенос выполняем. Если после переноса нарушение границ было устранено, то операцию балансировки завершаем. В противном случае переходим к следующей вершине;
 - (c) если после просмотра всех вершин устранить нарушение не удалось, завершаем работу процедуры $divide(V', k')$ аварийно.
8. Когда все вершины распределены по группам без нарушения границ Δ и δ , то необходимо последовательно произвести рекурсивные вызовы процедуры для каждой из полученных групп, указывая соответствующие значения V' и k' . Если какой-нибудь из них завершиться аварийно, то также завершаем работу аварийно.

Необходимо описать два частных случая при $k' = 2$ и $k' = 3$. Приведём сначала случай для $k' = 2$.

1. Пока есть нераспределённые вершины, и уровни загрузки каждой группы не достигли соответствующего этой группе значения δ , относим ближайшую вершину к той группе, уровень наполнения которой дальше отстоит от значения δ .
2. После выполнения предыдущего шага возможно превышение границ Δ и δ . Если это произошло, то необходимо выполнить операцию балансировки, как она была описана выше. Если она завершилась неуспешно, то завершаем работу процедуры аварийно.

3. Оставшиеся вершины распределяем по принципу близости к некоторой группе в следующем порядке: если ближайшая к первой группе вершина приводит к меньшему росту количества товара, чем аналогичная вершина для второй, то добавляем вершину к первой группе, и наоборот.

Случай для $k' = 3$ может быть представлен следующим образом.

1. Пока есть нераспределённые вершины, и уровни загрузки каждой группы не достигли соответствующего этой группе значения δ , относим ближайшую вершину к той группе, уровень которой дальше отстоит от значения δ .
2. После выполнения предыдущего шага возможно превышение границ Δ и δ . Если это произошло, то необходимо выполнить операцию балансировки, как она была описана выше. Если она завершилась неуспешно, то завершаем работу процедуры аварийно.
3. Все оставшиеся вершины относим ко второй группе.

Необходимо решить вопрос, как оценивать качество получаемых групп при выборе наилучшего порядка s_1 и s_2 . Существует несколько способов. Можно, например, решать для этого ЗК для всей группы вершин каким-либо алгоритмом, не требующим большого времени для работы. В вычислительном эксперименте использовалась оценка снизу длины возможного пути коммивояжёра, полученная путём суммирования длин рёбер минимального остова, построенного на полном графе вершин группы, с добавлением длины самого длинного ребра.

В целом работу алгоритма можно представить в следующем виде.

1. Вычисляем начальное значение количества экипажей $k = \lceil \sum_{i=1}^n Q(v_i) \rceil / q$. Если $k = 1$, то задача является ЗК в её обычной постановке, и нет необходимости проводить кластеризацию.
2. Произведём вызов процедуры $divide(V, k)$, передав ей в качестве параметров все множество вершин V и вычисленное текущее значение $k > 1$. Процедура может вернуть в виде результата множество групп, меньшее, чем k , это допустимое поведение. Если процедура завершила свою работу аварийно, то увеличиваем значение k на единицу и повторяем попытку до тех пор, пока результат не будет получен.

3. Вычисляем совокупность маршрутов, решая ЗК для вершин каждой из полученных групп. В вычислительном эксперименте использован один из лучших приближенных алгоритмов — алгоритм Лина-Кернигана [68].

Оценим трудоемкость алгоритма. При работе процедуры *divide* каждый из шагов 2, 3, ..., 7 имеет порядок трудоемкости не выше $O(n^2)$. С учетом рекурсивных вызовов на шаге 8 получаем рекуррентное соотношение для трудоемкости $T(n) = T(n/2) + c \cdot n^2$. Решив его, получаем величину общей трудоемкости $O(n^2)$. При выводе не было учтено, что выполнение процедуры *divide* может закончиться аварийно, и тогда все вычисления повторяются с самого начала. Если количество таких ситуаций не превысит константы, то трудоемкость будет иметь тот же порядок $O(n^2)$. Трудоемкость алгоритма Лина-Кернигана близка к $O(n^3)$, но так как этот алгоритм применяется по отдельности к каждой из k групп вершин, то общая трудоемкость на этой фазе алгоритма — $O((k \cdot n^3)/k^3) = O(n^3/k^2)$. Если с ростом n количество групп k растет пропорционально n , то трудоемкость 2-й фазы будет $O(n)$, а если при росте n количество групп k остается константой, то трудоемкость — $O(n^3)$. Таким образом, порядок общей трудоемкости всего алгоритма будет между $O(n^2)$ и $O(n^3)$.

2.12 Вычислительный эксперимент для ЗМТУГ

Качество предложенного алгоритма оценивалось путём постановки вычислительного эксперимента. Одни и те же наборы входных данных подавались на вход алгоритма Османа, описанного в работе [74], сбалансированного алгоритма и комбинированного алгоритма, когда на первом этапе выполняется сбалансированный алгоритм, а на втором — алгоритм Османа. В алгоритме Османа параметру, определяющему количество обмениваемых между маршрутами вершин, задано значение 1. Как отмечено в [74], хотя большее значение этого параметра и позволяет несколько повысить качество решения, однако при этом существенно возрастает время выполнения алгоритма. В алгоритме Османа имеется другой важный целочисленный параметр: длина списка исключений. Его наиболее предпочтительное значение вычисляется по эмпирической формуле (2).

В первой части вычислительного эксперимента алгоритмы испытывались на задачах (наборах данных), предложенных Кристофидесом, Мингоззи и

Тоссом [26], на которых испытывались также многие другие алгоритмы решения ЗМТ. Всех задач 14, из них выбраны те, которые соответствуют постановке ЗМТ с учетом грузоподъемности. Для получения более высокого качества решений алгоритм Османа выполнялся многократно с изменением параметра длины списка исключений в диапазоне от 7 до удвоенного значения, предложенного автором, и выбирался самый лучший из полученных результатов. Результаты вычислений, приведенные в табл. 5, содержат суммарные длины вычисленных алгоритмами маршрутов, количества получившихся при этом маршрутов (транспортных средств), а также общее затраченное время.

Из табл. 5 видно, что сбалансированный алгоритм уступает алгоритму Османа по качеству решения на шести примерах из семи, но затрачивает на вычисления в сотни раз меньше времени. Превосходство алгоритма Османа объясняется относительно небольшой размерностью задач, при росте количества вершин картина меняется в обратную сторону. В то же время, если решение, полученное сбалансированным алгоритмом, использовать как начальное для алгоритма Османа, то во всех случаях достигается наилучшее качество.

Во второй части вычислительного эксперимента алгоритмы испытывались на случайно сгенерированных наборах данных. Вершины в виде точек на плоскости размещались согласно равномерному распределению внутри единичного квадрата, депо помещалось в центре. Матрица стоимости переездов вычислялась в виде евклидовых расстояний между точками. Веса вершин (потребности в товаре) генерировались согласно дискретному равномерному распределению в диапазоне от 1 до 10. Для наборов данных из 50 вершин алгоритмы запускались по 100 раз, для 100 вершин — 50 раз, для 150 вершин — 25 раз и для 200 вершин — 10 раз. Результаты вычислений, приведенные в табл. 6, содержат отношения суммарных длин полученных маршрутов к их оценке снизу — сумме длин ребер минимального остова на всех вершинах и длины самого длинного ребра остова. В табл. 6 приведены минимальные, средние и максимальные величины этих отношений по всем выборкам, а также среднее время работы алгоритмов на одном наборе данных. При этом, в отличие от табл. 5, алгоритм Османа выполнялся только при одном значении параметра длины списка исключений, вычисляемом по формуле (2).

Из табл. 6 видно, что на первых четырех выборках, когда количество вершин в каждом из полученных маршрутов было в среднем менее, чем 10,

сбалансированный алгоритм оказался хуже алгоритма Османа по качеству решения в среднем от 0,5% до 3% при времени работы в десятки раз меньшем. На последних трех выборках, когда количество вершин в каждом из полученных маршрутов росло с ростом общего числа вершин, картина обратная, при этом преимущество сбалансированного алгоритма достигает более 10% при $n = 200$. При этом сбалансированный алгоритм затрачивает на вычисления в десятки раз меньше времени. Применение же комбинации этих двух алгоритмов всегда дает выигрыш (в среднем) по сравнению с алгоритмом Османа, который при $n = 200$ и грузоподъемности 200 доходит до 15% при заметном уменьшении времени вычислений.

Для вынесения решения о том, какой из алгоритмов оказался лучшим на той или иной случайной выборке, в табл. 7 приведены результаты сравнения алгоритмов между собой по качеству получаемых решений в таком виде, чтобы можно было применить знаковый статистический критерий. В каждой ячейке указывается, сколько раз из скольки алгоритм Османа проиграл по качеству сбалансированному или комбинированному алгоритму.

Применение критерия знаков к табл. 7 подтверждает преимущество сбалансированного алгоритма над алгоритмом Османа для последних трех выборок и преимущество комбинированного алгоритма над алгоритмом Османа для всех семи выборок. Согласно статистическим таблицам [2], это утверждение имеет вероятность ошибки менее 0,005.

Вычислительный эксперимент проводился на системе с процессором Intel Core 2 Duo E8400 под управлением 64-битного варианта GNU/Linux v2.6.25.

Таблица 5. Качество и время решения задач Кристофидеса, Мингоззи и Тосса. В ячейках таблицы приведены абсолютные значения длины решений, количество экипажей в решении и время работы. Достоверность значений равна 0,95 с двумя значащими цифрами после запятой.

Задача, кол-во вершин	Алг. Османа	Сбал. алг.	Комб. алг.
1 (50)	537,6 (5 эк.); 9 с	559,7 (5 эк.); 0,07 с	558,2 (5 эк.); 6 с
2 (75)	885,1 (11 эк.); 35 с.	937,9 (11 эк.); 0,11 с	881,6 (11 эк.); 26 с
3 (100)	867,2 (8 эк.); 161 с	1088,5 (8 эк.); 0,16 с	863,2 (8 эк.); 201 с
4 (150)	1122,5 (12 эк.); 993 с	1171,9 (12 эк.); 0,41 с	1078,9 (12 эк.); 682 с
5 (199)	1419,6 (17 эк.); 2749 с	1462,7 (17 эк.); 1,8 с	1379,4 (17 эк.); 3715 с
11 (120)	1223,7 (7 эк.); 227 с	1170,2 (7 эк.); 1,2 с	1059,2 (7 эк.); 203 с
12 (100)	905,2 (10 эк.); 220 с	1066,2 (10 эк.); 0,6 с	841,4 (10 эк.); 167 с

Таблица 6. Качество и время решения на случайных выборках. В ячейках указано минимальное, среднее и максимальное нормированное значение качества решения, а также среднее время работы алгоритма. Достоверность значений равна 0,95 с двумя значащими цифрами после запятой.

Число вершин, грузоподъемность	Алг. Османа	Сбалан. алг.	Комбин. алг.
50 (50)	1,60/1,83/2,17; 0,5 с	1,58/1,84/2,14; 0,04 с	1,46/1,73/2,10; 0,5 с
100 (50)	1,97/2,19/2,52; 3,6 с	1,96/2,23/2,56; 0,2 с	1,87/2,10/2,33; 2,7 с
150 (50)	2,16/2,43/2,68; 23 с	2,34/2,52/2,69; 0,5 с	2,17/2,37/2,53; 17 с
200 (50)	2,53/2,73/2,86; 87 с	2,60/2,83/2,99; 0,9 с	2,48/2,66/2,79; 58 с
100 (100)	1,53/1,73/1,99; 5,5 с	1,49/1,60/1,73; 0,3 с	1,40/1,51/1,67; 4,1 с
150 (150)	1,53/1,66/1,82; 33 с	1,42/1,52/1,63; 0,8 с	1,36/1,43/1,50; 19 с
200 (200)	1,51/1,62/1,79; 144 с	1,37/1,45/1,52; 2,1 с	1,33/1,38/1,43; 59 с

Таблица 7. Сравнение алгоритмов по качеству получаемых решений. В ячейках указано количество проигрышей из общего количества запусков.

Число вершин, грузоподъемность	Проигрыши алг. Османа	Проигрыши алг. Османа
	сбалан. алг.	комбин. алг.
50 (50)	45 из 100	89 из 100
100 (50)	17 из 50	45 из 50
150 (50)	4 из 25	23 из 25
200 (50)	2 из 10	10 из 10
100 (100)	45 из 50	50 из 50
150 (150)	24 из 25	25 из 25
200 (200)	10 из 10	10 из 10

Как показал вычислительный эксперимент, качество решений предложенного сбалансированного алгоритма в некоторых ситуациях незначительно уступает одному из лучших алгоритмов — алгоритму Османа, однако при увеличении количества вершин до 150 и более при пропорциональном увеличении вершин в отдельных маршрутах предложенный алгоритм начинает превосходить алгоритм Османа при одновременном уменьшении времени вычислений в десятки раз. Благодаря относительно невысокому порядку трудоемкости (менее $O(n^3)$) алгоритм можно применять для задач размерности до 1000 вершин и более, что недоступно алгоритму Османа.

Особенно эффективно применение предложенного алгоритма, дающего на-

чальное приближение для алгоритма Османа. Интересно, что при этом время работы алгоритма Османа на второй стадии заметно уменьшается. На практике применение комбинированного алгоритма удобно тем, что если время работы на второй стадии оказывается чрезмерным, процесс вычислений можно прервать и использовать полученный к этому моменту вариант решения задачи.

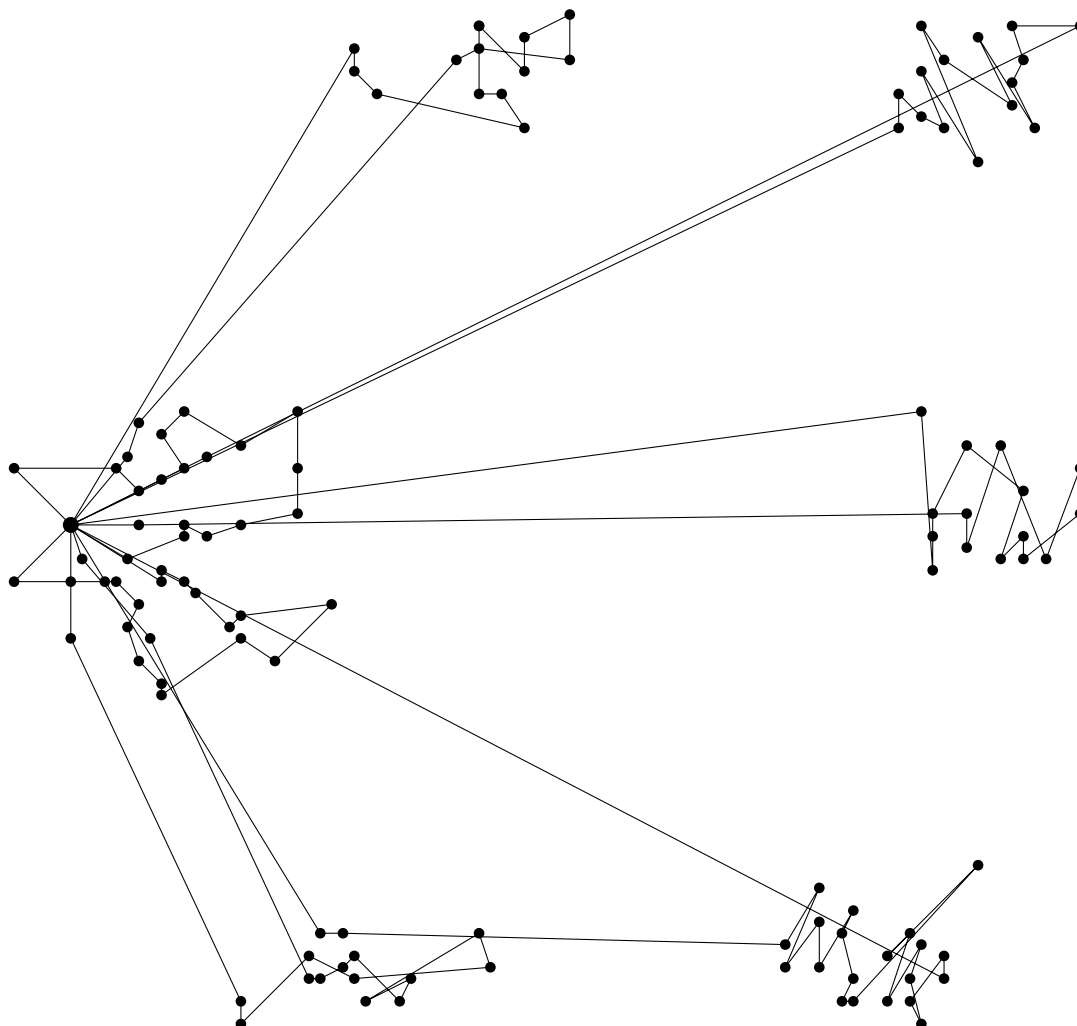


Рис. 9. Пример решения задачи Кристофидеса-Мингоззи-Тосса №11 алгоритмом Османа поиска с исключениями. Изображено семь маршрутов, проходящих через 120 вершин. Суммарная длина решения равна 1223 в условных единицах авторов задачи.

2.13 Дополнительные варианты сбалансированного алгоритма для ЗМТУГ

Выше был приведён вариант сбалансированного алгоритма, который показывал наиболее удачное сочетание всех характеристик своей работы. Тем не менее существуют некоторые его разновидности, которые могут оказаться

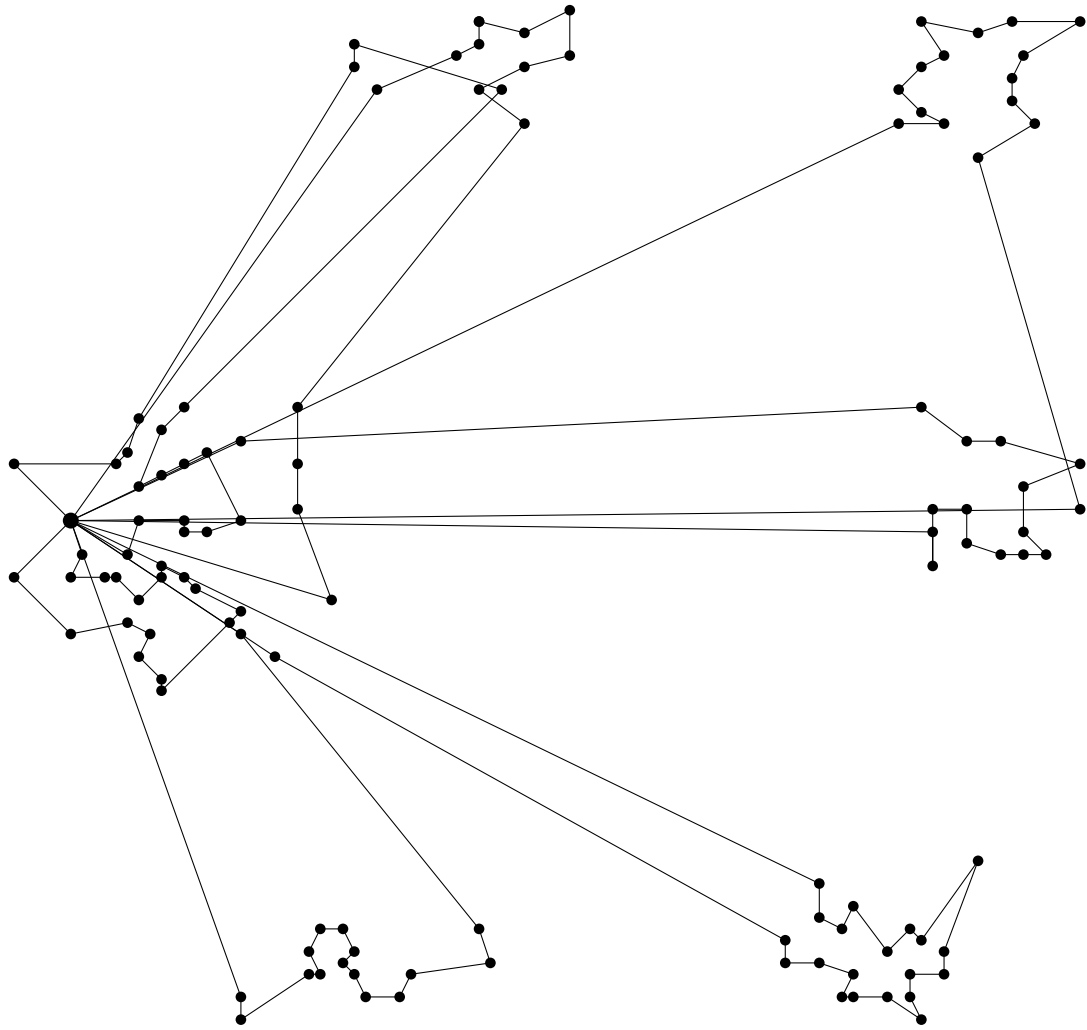


Рис. 10. Пример решения задачи Кристофидеса-Мингоzzi-Тосса №11 сбалансированным алгоритмом. Изображено семь маршрутов, проходящих через 120 вершин. Суммарная длина решения равна 1170 в условных единицах авторов задачи.

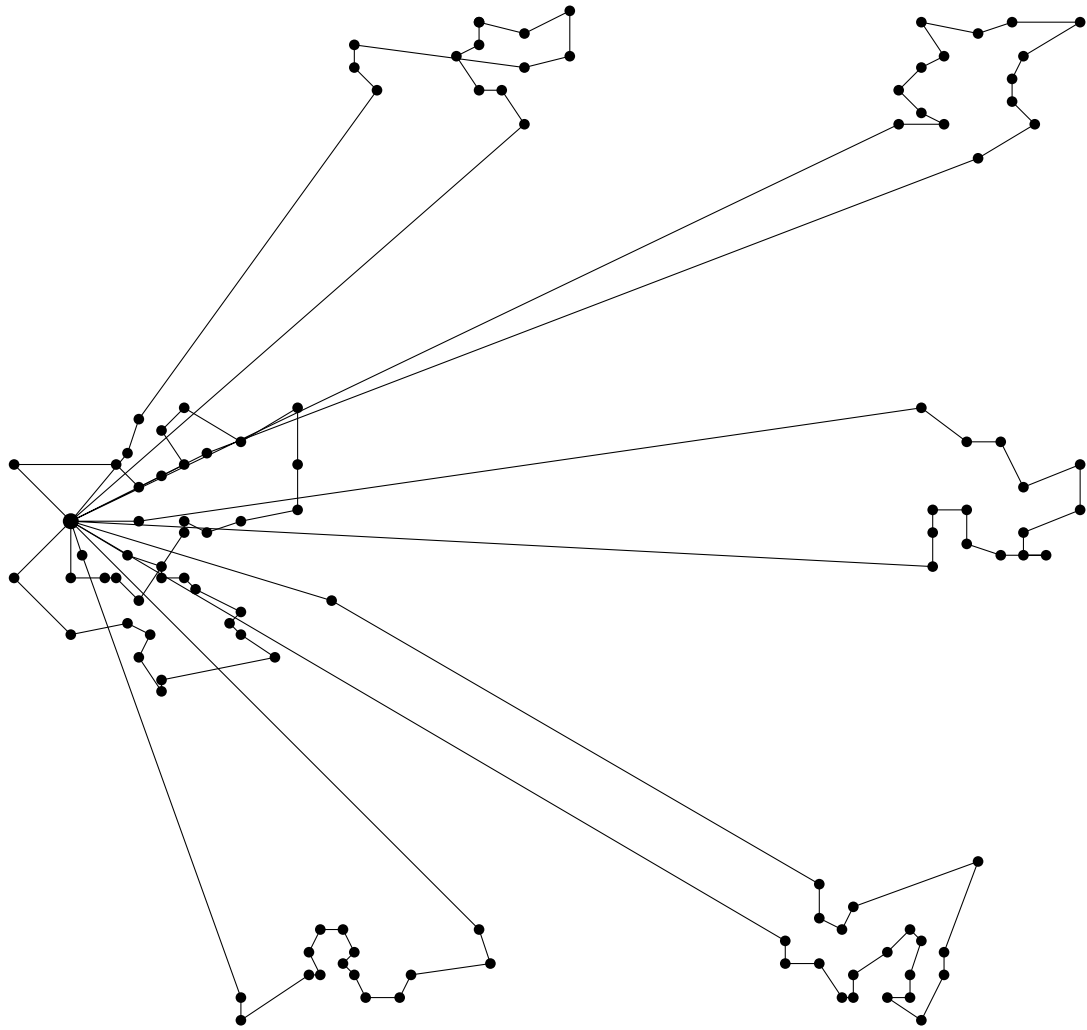


Рис. 11. Пример решения задачи Кристофидеса-Мингоzzi-Тосса №11 смешанным алгоритмом. Изображено семь маршрутов, проходящих через 120 вершин. Суммарная длина решения равна 1059 в условных единицах авторов задачи.

на практике полезны в условиях особых предпочтений. Приведём их описание без постановки вычислительного эксперимента, результаты которого должны отражать внесённые изменения.

2.13.1 Вариант сбалансированного алгоритма с бэктрекингом

Как говорилось в разд. 2.11, рекурсивная процедура $divide(V', k')$ имеет две величины Δ и δ , указывающих границы наполнения каждой из двух строящихся групп. В приведённом описании использовалось понятие “идеального” уровня наполнения групп, и тем самым отдавалось предпочтение равномерности загрузки транспортных средств, хотя любой вариант, укладывающийся в границы между δ и Δ , является допустимым по отношению принятого нами критерия сбалансированности. Поиск удачного отклонения от “идеального” распределения может дать более удачные решения с точки зрения стоимости переездов. Приведём модифицированный вариант алгоритма с возможностью бэктрекинга, который частично повторяет описание процедуры $divide(V', k')$ из разд. 2.11. Рассмотрение частных случаев при $k' = 1$ и $k' = 2$ опускаем.

1. Инициализация параметров. Переменная q_t должна содержать величину общей потребности в товаре для всех вершин из V' . Переменные k'_1 , k'_2 , Δ_1 , δ_1 , Δ_2 и δ_2 имеют такое же назначение, как и в первом варианте процедуры. Напомним, что они вычисляются по формулам:

$$\begin{aligned} k'_2 &= \lfloor \frac{k'}{2} \rfloor, \\ k'_1 &= k' - k'_2, \\ \Delta_1 &= k'_1 \cdot q, \\ \Delta_2 &= k'_2 \cdot q, \\ \delta_1 &= q_t - \Delta_2, \\ \delta_2 &= q_t - \Delta_1. \end{aligned}$$

Значения δ_1 или δ_2 могут оказаться отрицательными. В этом случае величина k' уменьшается на 1 и вычисления повторяются.

2. Далее производим поиск двух вершин s_1 и s_2 с максимальной стоимостью переезда между ними.

3. Далее на основе различных критериев относим оставшиеся вершины из V' в первую или вторую группу. Понятие близости вершины к группе оставляем без изменений.
4. Если $k'_1 \neq k'_2$, то выполняем добавление к первой группе ближайших к ней вершин до тех пор, пока не будет достигнута величина средней загрузки экипажа, которая вычисляется как q_t/k' .
5. Пока есть нераспределённые вершины относим поочерёдно их к первой и второй группе, выбирая каждый раз ближайшую.
6. Полученные уровни загрузки групп должны быть обязательно не меньше, чем δ и не больше, чем Δ , которые были вычислены выше (Δ_1, δ_1 для первой группы и Δ_2, δ_2 — для второй). Если какая либо группа нарушает это условие, то необходимо произвести процедуру балансировки:
 - (a) просматриваем вершины той группы, где произошло превышение Δ . Обратим внимание, что превышение этого значения у обеих групп одновременно невозможно. Порядок просмотра должен быть обратным тому, как происходило приписывание вершин группе;
 - (b) если перенос некоторой вершины в противоположную группу не приводит к превышению значения в ней, то такой перенос выполняем. Если после переноса нарушение границ было устранено, то операцию балансировки завершаем. В противном случае переходим к следующей вершине;
 - (c) если после просмотра всех вершин устранить нарушение не удалось, завершаем работу процедуры $divide(V', k')$ аварийно.
7. Итак, мы получили первое наполнение групп. Выполним отдельно для первой и второй группы последовательно следующие действия:
 - (a) просматриваем вершины в группе в порядке, обратном тому, как они присваивались ей с учётом процедуры балансировки;
 - (b) выполняем пробный перенос вершины в противоположную группу. Если перенос не нарушает границ Δ и δ , то производим рекурсивный вызов процедуры $divide(V', k')$ с получившимися наполнениями групп. Оцениваем качество возвращённых групп (одним из методом, описанных в разд. 2.11), если ранее уже были пробные рекурсивные

вызовы, то новое значение запоминаем при условии, что оно лучше прежнего.

- (с) возвращаем перемещённую вершину на её исходное место. Если все пробные вызовы завершились аварийно, то также завершаем работу аварийно.

Проведя серию рекурсивных вызовов процедура определит наиболее удачное расположение вершин, которое не будет нарушать наложенные границы балансирования. Очевидно, что при большом количестве вершин вызовов процедуры станет очень много, поэтому количество пробных вызовов на каждом шаге можно явно ограничить. Например, делаем всегда не больше двух или трёх попыток.

2.13.2 Ограничение количество вершин в маршрутах

Приведём ещё одну модификацию процедуры $divide(V', k')$, описанной в разд. 2.11, Особенность этого варианта заключается в том, что его можно применять для случаев с ограничением количества вершин в маршруте каждого транспортного средства. Другими словами, он приспособлен для решения ЗМТОКВ, описанной в разд. 1.2.

Задаётся величина $d > 0$ — максимальное количество вершин-клиентов для каждого построенного маршрута. Основное отличие заключается в том, что деление необходимо продолжить в тех случаях, когда условие ограничения грузоподъёмности уже выполнено, но ещё не выполнено условие ограничения количества вершин.

Процедура $divide(V', k')$ может возвращать разделение на группы, количество которых отличается от k' . Во всех ранее описанных вариантах количество групп могло получиться только меньше, чем k' , теперь же возможны случаи, когда количество групп будет превосходить k' .

Изменения затрагивают только ситуации при $k' = 2$ и $k' = 3$ (напомним, что они описывались отдельно в виде частных случаев). Если k'_1 или k'_2 получили значение, равное единице, то рекурсивное деление необходимо продолжить, если количество вершин в первой или второй группе, в зависимости от того, $k'_1 = 1$ или $k'_2 = 1$, превышает значение d .

При следующем вызове значение k' нужно задать равным $\lceil n/d \rceil$, где n — количество вершин, получившимся в группе для дальнейшего деления.

Обратим внимание, что при подобном рекурсивном вызове процедура $divide(V', k')$ должна также иметь несколько модифицированный вид. Это объясняется тем, что некоторые из чисел $\Delta_1, \delta_1, \Delta_2, \delta_2$ получатся отрицательными, т. к. суммарная потребность в товаре для вершин из V' намного меньше, чем грузоподъёмность k' транспортных средств. Таким образом, процедура деления получается значительно упрощённой.

1. Переменные k'_1 и k'_2 хранят значение, вершины для какого количества экипажей должны быть отнесены к первой и второй группам, они вычисляются по формулам:

$$k'_2 = \lfloor \frac{k'}{2} \rfloor,$$

$$k'_1 = k' - k'_2.$$

Обозначим через n_1 и n_2 количество вершин, которое должно быть отнесено к первой и второй группам. Вычислим их следующим образом:

$$n_2 = \lfloor \frac{n}{k'} \rfloor \cdot k'_2$$

$$n_1 = n - n_2,$$

где n — количество вершин в V' .

2. Далее ищутся две вершины s_1 и s_2 , с максимальной стоимостью переезда между ними. Вершины s_1 и s_2 являются начальным наполнением будущих групп. Если k'_1 не равно k'_2 , то приводимое ниже описание алгоритма необходимо выполнить два раза с обменом значений s_1 и s_2 и выбором наилучшего варианта. Вопрос, как сравнить качество двух вариантов, обсуждался в разд. 2.11.
3. Вершины s_1 и s_2 являются начальным наполнением будущих групп. Далее на основе различных критериев мы будем относить оставшиеся вершины из V' в первую или вторую группу. Понятие близости вершины к группе имеет такой же смысл, как и в разд. 2.11.
4. Если $k'_1 \neq k'_2$, то выполняем добавление к первой группе ближайших к ней $n_1 - n_2$ вершин.

5. Далее поочерёдно относим по одной вершине сначала к первой группе, затем ко второй. Каждый раз должна выбираться ближайшая вершина к нужной группе.
6. Если $k'_1 > 1$ или $k_2 > 2$ производим рекурсивный вызов процедуры передавая на вход соответствующие значения V' и k' .

В заключение описания обратим внимание, что подобный упрощённый вид процедуры деления на две группы не имеет исключительных ситуаций и всегда должен заканчиваться успехом.

2.14 Вариант сбалансированного алгоритма для нескольких депо

Рассмотрим способ применения сбалансированного алгоритма для случая с несколькими депо. Вариант ЗМТ для нескольких депо рассматривался в разд. 1.2. Приведём расширение постановки СЗМТ, описанной в разд. 2.1:

1. Пусть дано $n + t$ вершин (n вершин-клиентов и t вершин-депо), симметричная матрица стоимости переезда между всеми $n + t$ вершинами и число k — желаемое количество маршрутов.
2. В каждый результирующий маршрут должна включаться любая из t вершин-депо.
3. Каждая вершина-клиент входит только в один и только один результирующий маршрут.
4. Количество вершин для любой пары маршрутов не должно отличаться более, чем на единицу.
5. Суммарная стоимость объезда всех маршрутов должна быть минимальной.

Поскольку считаем, что каждое депо обладает неограниченным ресурсом товара и транспортных средств, приводимый алгоритм можно использовать не только для строго сбалансированного вида задачи, как мы сейчас описали, но и для общего вида ЗМТ для нескольких депо, описанного в разд. 1.2. Основное отличие заключается в том, что при явно заданном количестве транспортных средств k можно заранее определить, сколько вершин должно быть отнесено к каждому из депо. Если k не задано, то решить вопрос о количестве вершин-клиентов для каждого депо однозначно можно только в случае,

если общее количество вершин делится на количество депо нацело. Опустим рассмотрение вопроса о распределении вершин, если это условие не выполняется.

В предлагаемом алгоритме можно выделить три этапа:

1. Разделение множества вершин-депо и близким к ним вершин-клиентов на группы, в каждой из которых остаётся только одна вершина-депо, а количество вершин-клиентов соответствует определённому заранее для этого депо.
2. Разделение вершин-клиентов на группы для каждого транспортного средства (решение обычной ЗМТ).
3. Построение замкнутого маршрута по всем вершинам для каждого транспортного средства (решение ЗК).

Работу первого этапа можно представить в следующем виде:

1. Множество вершин-депо делится на два множества, отличающихся по количеству элементов не более чем на один. При этом минимальное расстояние между парами вершин-депо из разных множеств должно быть максимальным.
2. Все остальные вершины-клиенты делятся между этими двумя множествами с выбором по разности минимальных стоимостей переездов. Вычисление разности минимальных стоимостей должно проводиться аналогично тому, как это делалось в процедурах дихотомического деления, описанных в предыдущих разделах этой главы.
3. Если в одном из множеств содержится более одной вершины-депо, описанный алгоритм нужно повторить для неё ещё раз. Рекурсивные вызовы необходимо продолжать до тех пор, пока не останется по одной вершине-депо в каждой группе.

Если в каком-то из полученных множеств имеется более одного депо, то производится дальнейшее деление таким же способом.

Для оценки качества решений приведём некоторые результаты вычислительного эксперимента. В нём вершины генерировались как случайные точки с равномерным распределением внутри единичного квадрата, а расстояние вычислялось как евклидово. В табл. 8 и табл. 9 приведены отношения

суммарных длин всех полученных маршрутов к длине минимального остова по всем вершинам-клиентам и вершинам-депо.

Таблица 8. Результаты тестирования для двух депо. После наклонной черты в ячейках таблицы приведены для сравнения значения стоимости построенных маршрутов для одного депо. Достоверность значений равна 0,95 с двумя значащими цифрами после запятой.

Число маршрутов	16 вершин	32 вершины	64 вершин	128 вершин	256 вершин	512 вершин	1024 вершины
2	1,556/ 1,366	1,411/ 1,327	1,331/ 1,300	1,286/ 1,279	1,265/ 1,266	1,249/ 1,251	1,248/ 1,247
4	-	1,931/ 1,607	1,678/ 1,463	1,517/ 1,388	1,433/ 1,328	1,369/ 1,295	1,316/ 1,267
8	-	-	2,482/ 1,948	2,076/ 1,703	1,809/ 1,542	1,632/ 1,445	1,497/ 1,364
16	-	-	-	3,245/ 2,441	2,612/ 2,038	2,180/ 1,794	1,874/ 1,620
32	-	-	-	-	4,298/ 3,123	3,338/ 2,528	2,664/ 2,134
64	-	-	-	-	-	5,735/ 4,106	4,302/ 3,219
128	-	-	-	-	-	-	7,660/ 5,481

В случае двух депо они располагались по углам диагонали квадрата, а в случае четырёх депо — во всех четырёх углах. Количество вершин-клиентов задавалось в пределах от 16 до 1024, а число маршрутов — от 2 до 128. Для сравнения был рассмотрен случай одного депо, располагавшегося в центре квадрата, в таблицах численные результаты этого случая приведены после наклонной черты. Вычисление простых маршрутов коммивояжеров на третьем этапе производилось алгоритмом Лина-Кернигана.

2.15 Использование геометрической информации для процедуры деления вершин

В разд. 2.3 приведён общий вид процедуры дихотомического деления вершин на группы, использующий матрицу стоимостей переездов. Модификация этого метода для случая расположения вершин на плоскости может позволить значительно сократить время работы. Очевидно, что предположение о доступности такой информации является очень сильным допущением, но

Таблица 9. Результаты тестирования для четырёх депо. После наклонной черты в ячейках таблицы приведены для сравнения значения стоимости построенных маршрутов для одного депо. Достоверность значений равна 0,95 с двумя значащими цифрами после запятой.

Число маршрутов	32 вершины	64 вершин	128 вершин	256 вершин	512 вершин	1024 вершины
4	1,617/ 1,607	1,447/ 1,463	1,358/ 1,388	1,314/ 1,328	1,276/ 1,295	1,264/ 1,267
8	-	1,992/ 1,948	1,711/ 1,703	1,539/ 1,542	1,437/ 1,445	1,368/ 1,364
16	-	-	2,497/ 2,441	2,057/ 2,038	1,789/ 1,794	1,611/ 1,620
32	-	-	-	3,177/ 3,123	2,550/ 2,528	2,135/ 2,134
64	-	-	-	-	4,162/ 4,106	3,242/ 3,219
128	-	-	-	-	-	5,548/ 5,481

вершины-клиенты, которые должны быть обслужены транспортными средствами, на практике всегда располагаются на плоскости, и зачастую их географическое положение известно. В общем случае информация о расстоянии между вершинами на плоскости может достаточно сильно различаться с стоимостью переезда на транспортной сети, но обратим внимание, что геометрическая информация используется *только* для поиска пары максимально удалённых вершин, что с достаточной вероятностью должно соответствовать максимальной стоимости переезда между ними. Описанный ниже метод предлагается для использования в очень больших задачах, в которых количество вершин может исчисляться сотнями тысяч. При таких масштабах входных данных различия между расстоянием на плоскости и стоимостью переезда по транспортной сети должны сокращаться, следовательно эффективность метода будет возрастать.

Геометрический вариант процедуры предлагается только для начального деления вершин на группы, до тех пор, пока группы не будут содержать такое количество вершин, с которым сможет справиться стандартный подход из разд. 2.3, т. е., например, не более, чем 1000. Введём также ещё одно ограничение, часто выполняющееся на практике: предположим, что граф транспортной сети является разреженным. Другими словами, для каждой

его вершины количество вершин, смежной с ней, не превосходит некоторой величины k .

Формально все ограничения могут быть описаны следующим образом:

1. Все вершины располагаются на плоскости, и для каждой вершины i известны координаты $\{x_i, y_i\}$.
2. Задан разреженный неориентированный взвешенный связный граф транспортной сети, в котором из каждой вершины выходит не более, чем k рёбер, а вес рёбер соответствует стоимости переезда.

Задачу нахождения двух максимально удалённых вершин решим при помощи построения выпуклой оболочки. Построить такую оболочку можно, например, при помощи алгоритма “под-над” [13], в котором трудоёмкость определяется трудоёмкостью процедуры упорядочивания вершин, равной $O(n \log n)$. Построив выпуклую оболочку, выберем на ней вершину p с минимальной координатой x_i , и с минимальной координатой y_i , если их несколько. Далее требуется выполнить следующие действия:

1. От вершины p идём в любом направлении по выпуклой оболочке и вычисляем расстояние до каждой просмотренной вершины. На протяжении некоторого количества шагов расстояние должно увеличиваться, но в определённый момент произойдёт уменьшение. Вершину, в которой был достигнут максимум расстояния, обозначим q и остановим процесс.
2. Произведём перемещение вершины p по выпуклой оболочке в обратном направлении и будем продолжать эту работу до тех пор, пока расстояние между вершинами p и q увеличивается. Остановим работу в том положении, в котором было зафиксировано максимальное расстояние.

Трудоёмкость поиска вершин на выпуклой оболочке равна $O(n)$, следовательно, общее время поиска двух максимально удалённых вершин на плоскости может быть вычислено за время $O(n \log n)$.

Для дальнейшей работы необходимо вычислить кратчайшие пути на графе транспортной сети от вершин p и q до всех остальных вершин. Рассмотрим процесс для p , для q , он будет аналогичным.

1. Создадим множества вершин S и S' , поместим в S вершину p , а в S' — все вершины, смежные с p на графе транспортной сети. Кроме этого,

для всех вершин из S и S' необходимо поддерживать ссылку на некоторую вершину из S , позволяющую восстановить путь из каждой вершины до p . При начальном заполнении множества S' все ссылки должны указывать на p . Множество S' должно быть организовано по принципу пирамиды [4], позволяющей находить минимальный элемент с трудоёмкостью $O(\log n)$. В нашем случае минимальным элементом считается элемент с минимальной стоимостью переезда до вершины, на которую указывает соответствующая ему ссылка.

2. В цикле будем выполнять следующий шаг до тех пор, пока все вершины не попадут в S .
3. Найдём в S' вершину t , имеющую минимальное значение стоимости переезда до вершины, на которую указывается соответствующая ей ссылка. Исключим её из S' , поместим в S и добавим в S' все вершины, смежные с t . Причём, если некоторая вершина уже присутствует в S' , то необходимо обновить ссылку на t , если она будет соответствовать более дешёвому переезду, чем это было ранее.

Трудоёмкость описанных действий вычисляется следующим образом: последний пункт совершит ровно $(n - 1)$ шаг, где n — количество вершин, т. к. на каждом шаге добавляется по одной вершине. При добавлении вершины необходимо записать в пирамиду смежные вершины, количество которых не может превосходить k , а каждая запись может быть выполнена за логарифмическое время. Следовательно, общая трудоёмкость будет равна $O(nk \log n)$. Приведённый метод является модификацией алгоритма Дейкстры поиска кратчайших путей на графе [4].

Таким образом, мы можем получить минимальные расстояния от всех вершин до вершин p и q . Произведём распределение вершин по группам на основе разности стоимостей переездов, как это было описано в разд. 2.3, заменив поиск вершины с минимальной стоимостью переезда из текущего наполнения соответствующей группы постоянным использованием вершины p или q .

Поскольку процесс выполняется рекурсивной функцией, необходимо произвести оценку общей трудоёмкости всех рекурсивных вызовов. Это можно сделать путём вычисления рекуррентной формулы $T(n) = 2T(n/2) + cn \log n$ равной $T(n) = c'n \log^2 n$.

2.16 Выводы

Предлагаемый новый вид ЗМТ с дополнительным условием сбалансирования выглядит следующим образом [7, 12]:

1. Необходимо построить заданное заранее число замкнутых маршрутов, проходящих через определённое множество целевых вершин. Через каждую вершину должен проходить только один маршрут.
2. Все маршруты должны проходить через депо.
3. Количество вершин для каждой пары построенных маршрутов не должно отличаться не более, чем на единицу.
4. Суммарная стоимость объезда маршрутов должна быть минимальной.

Задача в такой формулировке названа сбалансированной ЗМТ. На примерах алгоритмов заметания и разрезания общего маршрута показано влияние условия сбалансирования на алгоритмы для ЗМТ в общем виде, приводящее к снижению объёма вычислений.

Предложена сбалансированная стратегия дихотомического разделения вершин на группы для каждого транспортного средства [7, 12], успешно адаптированная для трёх типов задач:

1. Сбалансированной ЗМТ [8].
2. ЗМТ с учётом грузоподъёмности [11].
3. ЗМТ для нескольких депо [9].

Для определения эффективности использования сбалансированной процедуры дихотомического деления проведён вычислительный эксперимент, сравнивающий новый алгоритм с алгоритмом Османа поиска с исключениями. Новый алгоритм показывает хорошие результаты для задач с количеством вершин более, чем 150, при заметно меньшем времени исполнения.

Предложен вариант процедуры дихотомического деления вершин для группы с использованием геометрической информации и для разреженных графов транспортной сети, способный решать задачу со временем $O(n \log^2 n)$.

3 Реализация алгоритмов решения ЗМТ

Разработка специализированных программных пакетов, приспособленных для повседневного применения в логистических компаниях, является завершающей стадией работы, позволяющей внедрить научные теоретические достижения на практике с получением экономического эффекта.

3.1 Основные понятия

Построение маршрутов производится с использованием информации о транспортной сети города или региона, позволяющей определить стоимость переездов между всеми требуемыми узлами. Такими узлами могут быть точки розничной торговли, клиенты компании, депо и пр. Функции построения и редактирования карты и транспортной сети являются стандартными функциями геоинформационных систем (ГИС), используемыми для решения ряда задач, в том числе и ЗМТ.

Основные этапы решения задачи построения маршрутов:

1. Загрузка основных данных карты из так называемого share-файла.
2. Обработка данных share-файла и построение транспортного слоя на карте.
3. Редактирование данных карты для учёта текущей обстановки на улицах города.
4. Построения графа транспортной сети на основе данных транспортного слоя карты.
5. Выбор пунктов обслуживания (клиентов для объезда), депо, указание дополнительных атрибутов решения задачи.
6. Вычисление матрицы стоимости переездов.
7. Запуск алгоритмов решения ЗМТ для вычисления маршрутов.
8. Сохранение результатов.

Основные материалы карты транспортных магистралей города часто хранятся в так называемом share-файле. Формат этого файла был разработан компанией Esri и содержит набор геометрических примитивов для представления элементов транспортной сети с их атрибутами. Большей частью это

вершины, полилинии и полигоны. Несмотря на то, что хранимая информация представляет из себя семантику графа, по структуре данных она графом не является. В файл записывается множество разрозненных объектов.

При открытии share-файла в системе создаётся новый слой на карте, предназначенный для построения и обработки транспортной сети. При этом хранимые примитивы объединяются в единую структуру, разработанную для эффективного выполнения операций редактирования пользователем и различных расчётов. Информация о транспортной сети может содержать объекты следующих типов:

1. **Узлы** — точечные объекты, отмечающие на карте положения различных пунктов. Для карты дорог узлами могут быть не только перекрёстки, но и просто важные для пользователя точки. Также узлами являются начала и концы дорог.
2. **Автомобильные дороги** — элементы карты, имеющие некоторую протяжённость. С автомобильными дорогами как правило ассоциирована некоторая дополнительная информация: ширина дороги, количество полос, ограничения скорости и пр.
3. **Дуги** — элементы, имеющие некоторую протяжённость, обозначающие возможность перевозки, но не являющиеся реальными дорогами (например, паромная переправа).
4. Другие элементы, такие как маршруты, кварталы, премыкания, используемые для решения разнообразных задач, но не востребованные при решении ЗМТ.

Со всеми элементами транспортной сети обычно связываются различные дополнительные данные. Они являются очень важными, т. к. некоторые атрибуты непосредственно участвуют в процессе расчёта матрицы стоимостей переездов. Дороги имеют определённое количество полос, ограничения скорости, могут позволять движение в обе или только одну сторону и пр. Эти детали обязательно учитываются в будущем при вычислении стоимости проезда. Аналогично, узлы, в которых соединяются две и больше дороги, хранят информацию о поворотах, разрешённых на перекрёстке.

При загруженном слое транспортной сети на карте пользователь может производить манипуляции с объектами, подбирая наилучшее представление

обстановки на дорогах города. Тем не менее, транспортная сеть непригодна для использования в алгоритмах решения различных задач. По ней невозможно правильно вычислить матрицу стоимостей переездов. Каждый перекрёсток может иметь различные правила поворотов на нём и другие ограничения.

Для получения структуры данных, пригодной для использования в алгоритмах, производится построение ориентированного взвешенного графа транспортной сети. Для этого выполняются следующие действия:

1. Каждый узел, на котором соединяются n дорог, заменяется $2n$ вершинами, из которых n вершин представляют возможность въезда на этот узел, и ещё n — выезда с него. Если некоторая дорога имеет возможность только въезда или только выезда, соответственно, оставляется только одна вершина.
2. Все полученные на предыдущем шаге вершины попарно соединяются дугами, чтобы правильно представить возможность переезда с одной дороги узла на другую. Если некоторый поворот запрещён, то соответствующая дуга пропускается. Дугам можно назначить некоторые стоимости, которые в большинстве случаев должны быть равными нулю, но могут иметь значение и отличное от нуля, если некоторый поворот возможен, но, например, вследствие сложной обстановки на него уходит много времени.
3. Все дороги транспортной сети заменяются парой дуг, представляющей движение в обе стороны по этой дороге. Если дорога разрешает движение только в одном направлении, то дуга добавляется в граф только одна. Таким же образом две дуги могут иметь различные отметки стоимости переезда.

Очевидно, что представление графа транспортной сети для запуска алгоритмов требует значительно больше памяти, чем это нужно просто для хранения карты дорог на транспортном слое карты. Если производится обработка карты большого города, то необходимо уделять серьёзное внимание ресурсам вычислительной системы.

После построения графа элементы действительности в различном виде представлены в математической модели, и все необходимые структуры готовы для запусков алгоритмов. Следующим этапом для решения ЗМТ является выбор целевого множества пунктов обслуживания и вычисление матрицы

стоимости переездов. Выбор пунктов рационально осуществлять уже после построения графа, т. к. пользователю может потребоваться запускать алгоритм несколько раз с разными наборами входных данных. Повторное выполнение процедуры построения графа приведёт к излишним затратам времени.

Матрица стоимости должна быть несимметричной и хранить информацию о переездах только между узлами, непосредственно обрабатываемыми при решении ЗМТ. Это должны быть узлы, где располагаются депо и пункты обслуживания. Обрабатывать все узлы, такие как, например, перекрёстки, необходимости нет. Это приведёт к излишним затратам памяти.

Дальнейшая работа сводится к вызову функций с реализацией алгоритмов решения ЗМТ, про которые рассказывается в разд. 3.4. Они возвращают информацию о построенных маршрутах, представленную в виде последовательности номеров вершин. Далее пользователь может оценить качество полученных решений, пересчитать их при необходимости с другими параметрами или сохранить результаты в виде отчётов, если они его устраивают.

3.2 Обработка несимметричных матриц

Используемые алгоритмы, кроме метода ветвей и границ решения ЗК, неспособны работать с несимметричными матрицами стоимости переездов. Рассмотрим некоторую оценку погрешности вычислений при переходе от несимметричной матрицы к симметричной путём выбора минимального значения среди пар элементов матрицы, расположенных симметрично друг от друга относительно главной диагонали. В алгоритмах, которые не способны работать с несимметричной матрицей стоимости, используется симметричная матрица, и после получения маршрутов выполняется их пробный разворот с целью определения наилучшего направления движения на основе значений несимметричной матрицы.

Покажем, что если при таком подходе совершается переход к симметричной матрице путём выбора минимального значения стоимости двух направлений движения, и каждое значение несимметричной матрицы не более, чем в $1 + \varepsilon$ раз больше соответствующего значения симметричной матрицы, то общая стоимость маршрута при обратном переходе к несимметричной матрице ухудшается не более, чем в $1 + \varepsilon/2$ раз. Если известно, что при этом также соблюдается правило треугольника, то будем называть такой случай *почти метрическим пространством расстояний* с точностью ε .

Рассмотрим наихудший вариант, когда каждое значение несимметричной матрицы либо равно соответствующему значению симметричной, либо ровно в $1 + \varepsilon$ раз больше него. Пусть l_i — стоимость i -того переезда в некотором маршруте на основе симметричной матрицы, а L — стоимость всего маршрута на основе значений симметричной матрицы. Числа q_i могут принимать значения либо равное нулю, либо равное ε , в зависимости от того, какое значение попало в симметричную матрицу. Таким образом, стоимость маршрута после перехода к несимметричной матрице L' равна:

$$\begin{aligned} L' &= l_1(1 + q_1) + \dots + l_n(1 + q_n) \\ L' &= (l_1 + l_1q_1) + \dots + (l_n + l_nq_n) \\ L' &= (l_1 + \dots + l_n) + (l_1q_1 + \dots + l_nq_n) \\ L' &= L + (l_1q_1 + \dots + l_nq_n) \end{aligned}$$

Выражение $l_1q_1 + \dots + l_nq_n$ обозначим через Q и рассмотрим его: числа q_i указывают, какие именно дуги имеют стоимость больше, чем аналогичная дуга в обратном направлении. Если произвести разворот маршрута, то такие q_i , которые равны ε , должны стать равными нулю и наоборот. Если все q_i были бы равны ε , то выражение приобрело бы вид $(l_1 + \dots + l_n)\varepsilon = L\varepsilon$. Таким образом, если имеем некоторое значение выражения Q , то новый вид после разворота можно записать как $L\varepsilon - Q$. Учитывая, что $Q \leq L\varepsilon$, мы всегда можем выбрать из выражений Q и $L\varepsilon - Q$ такое, которое меньше или равно $L\varepsilon/2$. Следовательно:

$$\begin{aligned} L' &\leq L + L\frac{\varepsilon}{2} \\ L' &\leq L\left(1 + \frac{\varepsilon}{2}\right) \end{aligned}$$

Из приведённого доказательства следует, что если есть возможность разворота маршрута, построенного на симметричной матрице, то при возврате к исходной матрице, результат станет дороже не более, чем в $1 + \varepsilon/2$ раз.

3.3 Пользовательский интерфейс

Подготовленная библиотека интегрирована в крупную геоинформационную систему IndorGIS. Опишем основные детали интерфейса пользователя, необходимые в работе при построении маршрутов объездов пунктов обслуживания.

1. **Дерево слоёв** — древовидный элемент управления, расположенный по умолчанию в левой части окна приложения (рис. 12). Предназначен для манипуляции со слоями карты. Позволяет выполнять следующие основные операции:
 - (а) создавать новые слои (транспортные, точечные, слои препятствий и др.);
 - (б) выбирать активный слой для редактирования;
 - (с) включать и отключать активные слои
 - (d) редактировать параметры слоя при помощи специального окна параметров.
2. **Контейнер карты**, главная функция которого — позволять редактировать содержимое карты. Контейнер позволяет в удобной форме создавать, изменять и удалять различные объекты на карте. Обычно занимает основное рабочее пространство окна приложения (рис. 13 и рис. 14).
3. **Инспектор объектов** позволяет задавать все доступные для изменения значения параметров активного объекта на карте. Такими параметрами могут быть название, ширина дороги и пр.
4. **Диалог настройки транспортной сети** позволяет определить, какой слой будет использоваться для транспортных расчётов, поскольку одна и та же карта может содержать несколько транспортных сетей (рис. 15). Кроме того, здесь настраиваются различные параметры транспортной сети.
5. **Диалог расчёта кратчайшего обхода** — основное окно выбора параметров построения замкнутых маршрутов (рис. 16). Оно позволяет выполнить следующие действия:
 - (а) выбрать тип задачи: построение только одного маршрута (решение ЗК), построить несколько маршрутов с явным указанием их количества или построение нескольких маршрутов с автоматическим определением их количества на основе данных о грузоподъёмности транспортных средств;
 - (б) выбрать желаемое качество решения, имеющего десять градаций от максимально быстрого метода до максимально качественного;

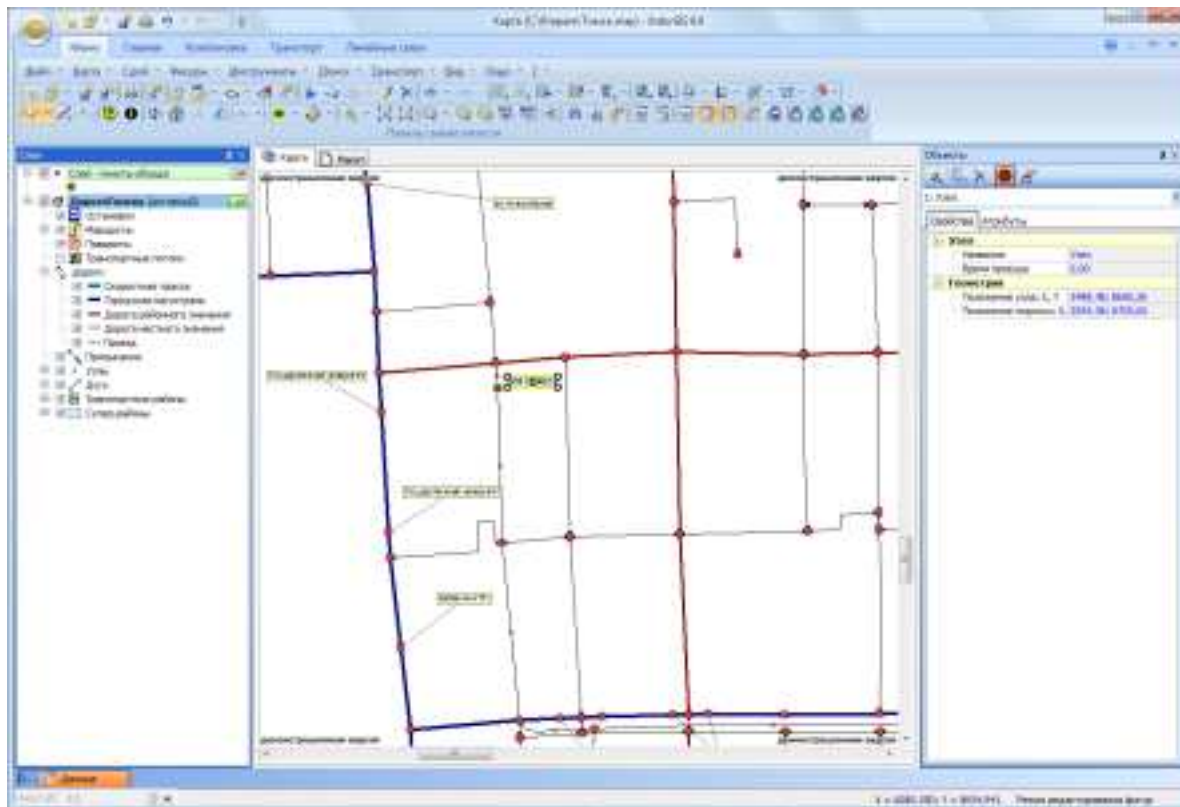


Рис. 12. Окно просмотра транспортной сети

- (с) указать количество маршрутов или грузоподъемность транспортных средств (потребность товара указывается на карте в виде параметров пунктов обслуживания);
- (d) запустить процесс вычислений.

При отображении построенных маршрутов можно включить опцию отображения со смещением, чтобы предотвратить наложение на экране и упростить просмотр.

3.4 Интерфейс библиотеки для решения ЗМТ

Реализация алгоритмов для решения ЗМТ вынесена в отдельную DLL-библиотеку, подключаемую к основному приложению. Библиотека экспортирует две функции, доступные для использования. На языке C++ их заголовки могут быть представлены в следующем виде:

- `int SolveBCVRP(size_t itemCount,`
`const double* rawMatrix,`
`size_t depoIndex,`
`size_t desiredVehicleCount,`

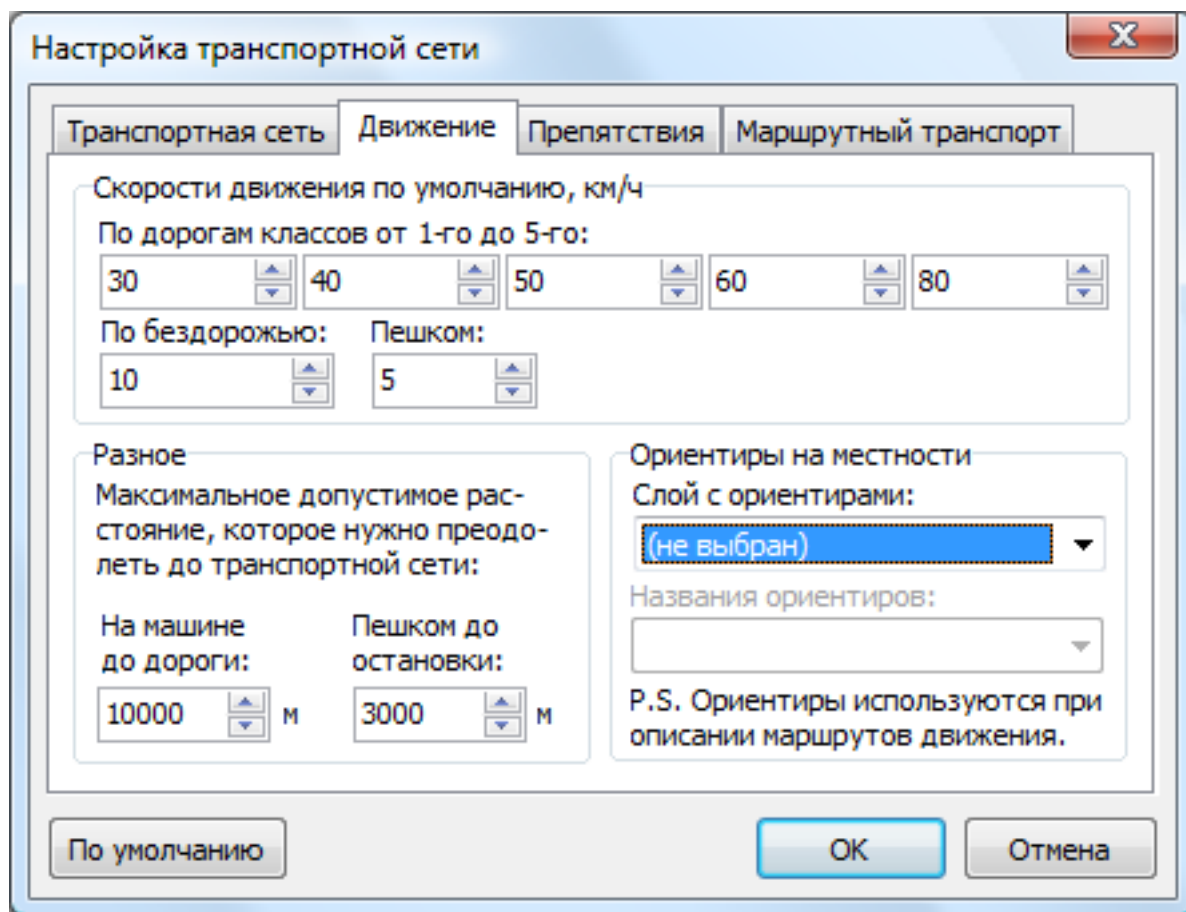


Рис. 15. Окно выбора параметров построения маршрутов

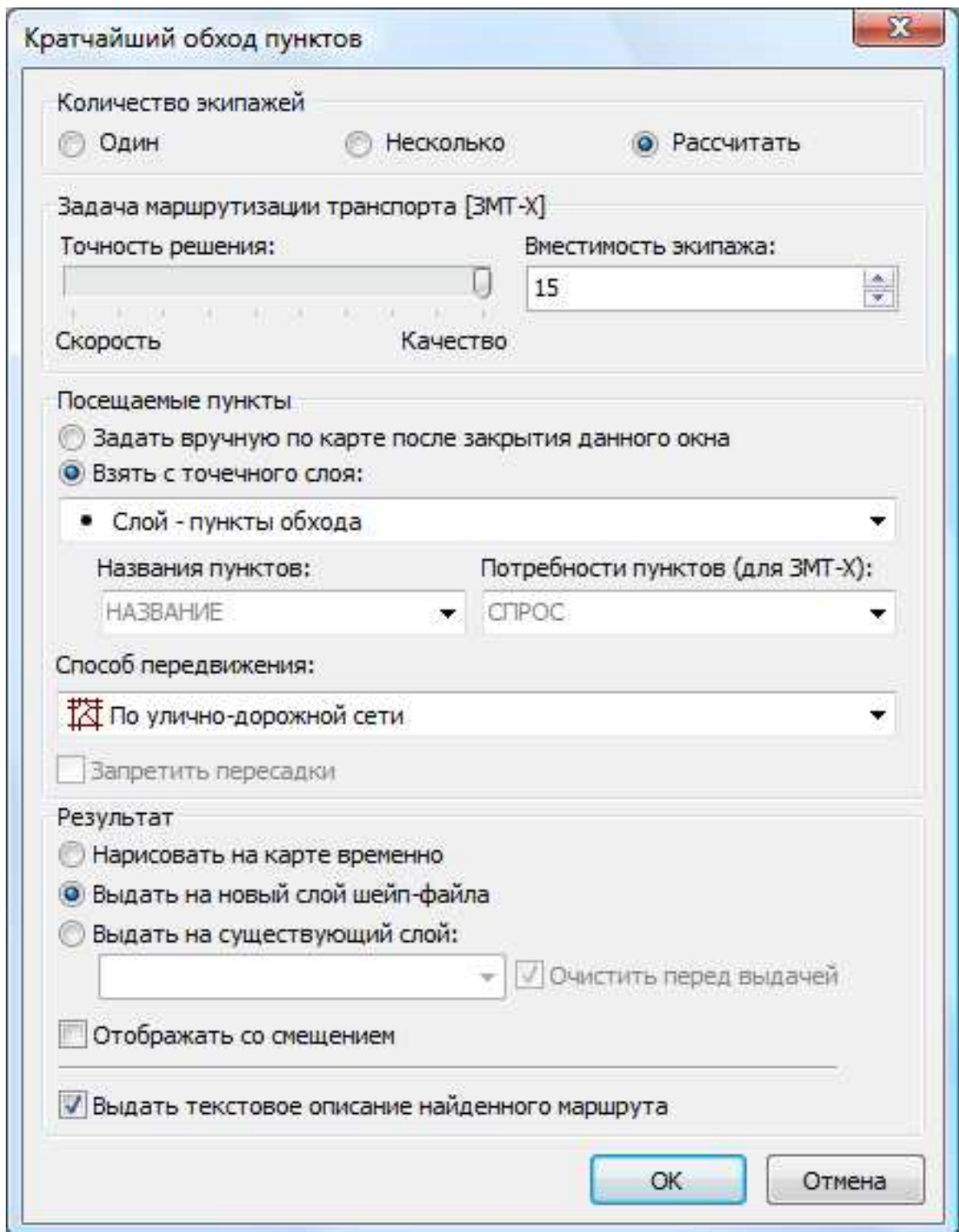


Рис. 16. Окно настроек транспортной сети

```
size_t desiredQuality,  
size_t* result);
```

- `int SolveVRP(size_t itemCount,
const double* rawMatrix,
size_t depoIndex,
size_t desiredQuality,
size_t vehicleCapacity,
const size_t* quantities,
size_t maxResultLen,
size_t* result);`

Функция `SolveBCVRP` используется для случаев, когда количество транспортных средств указывается явно, а их грузоподъемность не учитывается. Во всех маршрутах количество вершин не должно отличаться более, чем на единицу. Отсюда название: BCVRP — Balanced Cargo Vehicle Routing Problem. Функция `SolveVRP` решает ЗМТУГ в обычном виде, в котором количество транспортных средств вычисляется автоматически на основе информации о количестве товара для каждого клиента и грузоподъемности транспортных средств.

Обе функции возвращают код завершения операции. Он может принимать следующие значения:

- 0 — задача успешно решена.
- 1 — недопустимое значение аргумента.
- 2 — матрица стоимости несимметрична (зарезервировано, в настоящий момент не используется).
- 3 — матрица стоимости содержит отрицательные значения.
- 4 — матрица стоимости содержит ненулевые значения на главной диагонали.
- 5 — внутренняя ошибка библиотеки, детальная информация недоступна.
- 6 — в буфере для получения результата недостаточно места.

3.4.1 Описание функции SolveBCVRP

Как говорилось выше, функция `SolveBCVRP` используется в случаях, когда количество маршрутов задано заранее, а информация о потребности в товаре клиентов отсутствует. Функция принимает следующие параметры:

1. `size_t itemCount` — общее количество вершин, включая депо.
2. `const double* rawMatrix` — матрица стоимости поездок. Передаётся сначала первая строка матрицы, затем вторая и т. д. Матрица должна иметь размеры $itemCount \times itemCount$.
3. `size_t depoIndex` — индекс вершины-депо среди всего множества вершин.
4. `size_t desiredVehicleCount` — желаемое количество маршрутов.
5. `size_t desiredQuality` — желаемый уровень качества. Задаётся числом от 1 до 10.
6. `size_t* result` — указатель на буфер для получения результата. Буфер должен иметь длину $2 \cdot desiredVehicleCount + itemCount - 1$. В буфер в начале помещаются количества вершин в каждом результирующем маршруте, затем индексы вершин первого маршрута, затем индексы вершин второго маршрута и т. д. Проверка размера буфера внутри функции не выполняется.

Функция выбирает алгоритмы для решения задачи в зависимости от указанного уровня качества обработки. Обрабатываются следующие значения:

- 1 — сначала производится решение ЗК на всём множестве вершин, включая депо, используя алгоритм 2-опт [67], затем полученный маршрут разрезается на фрагменты для каждого транспортного средства с добавлением депо (см. разд. 2.7). Вариант для максимально быстрого получения некоторого решения.
- 2 — сначала производится решение ЗК на общем множестве вершин, включая депо, путём применения алгоритма ветвей и границ [69], если множество вершин содержит не более 16 элементов, и алгоритмом дерева [61] в противном случае. Затем выполняется разрезание полученного маршрута на фрагменты для каждого транспортного средства.

- 3 — аналогичен уровню качества 2, но каждый получившийся маршрут проходит дополнительную обработку: если в нём не более 16 вершин, то он обрабатывается методом ветвей и границ, в противном случае запускается алгоритм дерева с последующим пятикратным запуском алгоритма 2-опт и локальной оптимизацией с окном в девять вершин [61].
- 5–6 — множество вершин, исключая депо, сначала разделяется на группы при помощи сбалансированной процедуры дихотомической кластеризации (см. разд. 2.4), затем для каждой группы решается ЗК следующим образом: если в группе менее 16 вершин, то задача решается при помощи метода ветвей и границ, в противном случае при помощи алгоритма дерева, с последующим пятикратным запуском алгоритма 2-опт и локальной оптимизацией с окном в девять вершин.
- 7–8 — аналогичен предыдущему варианту, но отличается способом решения ЗК для групп после разделения вершин: если количество вершин в некоторой группе не больше 16, то запускается метод ветвей и границ, в противном случае используется алгоритм Лина-Кернигана (см. разд. 1.6.1) с последующим запуском локальной оптимизации с окном в девять вершин.
- 9–10 — аналогичен предыдущему варианту, но всегда используется метод ветвей и границ для решения ЗК без зависимости от количества вершин в группе.

Количество вариантов качества решения больше, чем количество принятых наборов алгоритмов, интервал значений от 1 до 10 выбран по соображениям будущей совместимости. Числовой параметр качества обработки должен скрывать от пользователя любую информацию об алгоритмах. Предполагается, что оператор не владеет информацией о них.

При запуске метода ветвей и границ для решения ЗК всегда используется несимметричная матрица стоимости переездов. Все прочие алгоритмы работают с симметричной матрицей, полученной из несимметричной путём выбора наименьшего значения. После получения маршрута выполняется его пробный разворот с целью определить наилучшее направление движения с учётом значений несимметричной матрицы.

3.4.2 Описание функции SolveVRP

Функция `SolveVRP` разработана для решения ЗМТУГ в общепринятой постановке, т. е. с указанием грузоподъёмности транспортных средств и потребности в товаре каждого клиента. Количество маршрутов вычисляется автоматически. Функция принимает следующие параметры:

1. `size_t itemCount` — общее количество вершин, включая депо.
2. `const double* rawMatrix` — матрица стоимости переездов. Передаётся сначала первая строка матрицы, затем вторая и т. д. Матрица должна иметь размеры $itemCount \times itemCount$.
3. `size_t depoIndex` — индекс вершины-депо в общем множестве вершин.
4. `size_t desiredQuality` — желаемый уровень качества обработки, задаётся числом от 1 до 10.
5. `size_t vehicleCapacity` — целое число больше нуля, представляющее значение грузоподъёмности каждого транспортного средства.
6. `const size_t* quantities` — массив целых чисел больше нуля, задающий потребности в товаре для всех клиентов. Элемент, соответствующий депо, должен иметь значение, равное нулю.
7. `size_t maxResultLen` — максимальное количество элементов, которое функция может заполнить в выходном буфере. Если длина буфера недостаточна, функция возвращает ошибку.
8. `size_t* result` — буфер для получения результата. Формируется следующим образом: сначала записывается число маршрутов, затем несколько чисел, указывающих количество вершин в каждом маршруте, затем индексы вершин первого маршрута, затем индексы вершин второго маршрута и т. д. Поскольку количество маршрутов высчитывается автоматически, определить заранее требуемую длину буфера невозможно.

Функция обрабатывает указания качества следующим образом:

- 1–2 — решение первоначально строится алгоритмом Кларка-Райта (см. разд. 1.4.1), после чего каждый построенный маршрут дополнительно обрабатывается при помощи метода ветвей и границ, если в нём не более 16 вершин, или при помощи алгоритма 2-опт в противном случае.

- 3–5 — сначала производится кластеризация для всего множества вершин, исключая депо, используя процедуру дихотомической кластеризации, описанной в разд. 2.11, после чего для каждой группы решается ЗК следующим образом: если в группе не более 16 вершин, то используется метод ветвей и границ, в противном случае запускается алгоритм Лина-Кернигана с последующим пятикратным запуском алгоритма 2-опт и локальной оптимизации с окном в девять вершин.
- 6–8 — задача решается алгоритмом Османа (см. разд. 1.8), используя в качестве начального решения маршруты, построенные алгоритмом Кларка-Райта. Далее проводится дополнительная оптимизация для каждого маршрута путём запуска метода ветвей и границ, если в нём не более 16 вершин, или алгоритмом 2-опт в противном случае.
- 9 — задача решается при помощи смешанного алгоритма, который из себя представляет алгоритм Османа, использующий в качестве начального решения маршруты, построенные дихотомической процедурой кластеризации с решением ЗК алгоритмом Лина-Кернигана. Как и в предыдущем случае для получения окончательного решения маршруты проходят дополнительную оптимизацию при помощи запуска метода ветвей и границ, если в маршруте не более 16 вершин, или при помощи алгоритма 2-опт в противном случае.
- 10 — аналогичен предыдущему варианту, но для получения окончательных маршрутов всегда используется метод ветвей и границ.

При запуске метода ветвей и границ всегда используется несимметричная матрица. В противных случаях матрица приводится к симметричной, и выполняется пробный разворот маршрутов для выбора наиболее удачного направления движения.

3.5 Внутренняя структура библиотеки алгоритмов решения ЗМТ

Библиотека алгоритмов решения ЗМТ создана на языке C++, имеет размер около 8200 строк исходного кода и содержит 70 классов различного назначения. Особое внимание уделено сохранению возможности конструировать на основе созданных классов приложения, способные решать более широкий круг задач, не предусмотренных на этапе проектирования.

Реализованные алгоритмы решения ЗМТ не требуют большого количества оперативной памяти для вычислений. В задачах, использующих матрицу стоимости переездов, память в основном расходуется на хранение данных о затратах на перемещение транспортных средств. Поскольку алгоритмы на основе матрицы стоимости применяются для обработки задач, содержащих около 1000 вершин, легко подсчитать, что для них потребуется около 8 МБ свободного пространства. Для геометрических задач, которые могут содержать до 1000000 вершин, матрица не требуется, и основной расход приходится на хранение самих вершин, что требует порядка 20 МБ свободного пространства. Приведённые оценки расхода оперативной памяти в десятки раз меньше, чем её объём, доступный в современных вычислительных устройствах, и нет необходимости применять дополнительные оптимизации.

Следующие концептуальные решения были приняты при реализации библиотеки:

1. Все типы вершин и других элементарных объектов должны обладать одинаковым набором методов и операторов, позволяющим их лёгкую взаимозамену.
2. Каждый алгоритм должен быть реализован в виде отдельного параметризованного класса (template), в котором через параметры обязательно передаётся тип вершины и тип других элементов (графов, дуг и пр.) при необходимости.
3. В реализации алгоритмов без использования геометрической информации между вершинами разрешены только операции присваивания и сравнения на равенство/неравенство. Получение каких-либо метрик напрямую без применения объектов-пространств запрещено (см. ниже).
4. В алгоритмах, использующих геометрическую информацию, разрешено получение вектора координат вершины в n -мерном пространстве, а также разрешены операции, описанные в пред. пункте.
5. В алгоритмы, требующих получение метрик между вершинами, должна быть возможность передать ссылку на объект-пространство, определяющий порядок вычисления стоимости переезда между двумя вершинами.
6. Все объекты-пространства должны обладать строго одинаковым набором методов, позволяющих выполнять следующие операции:

- получение стоимости переезда между двумя заданными вершинами;
- получение стоимости переезда, соответствующей некоторому ориентированному или неориентированному ребру, соединяющему две вершины;
- получение стоимости переезда, соответствующей проезду по замкнутому маршруту, заданному определённым количеством вершин.

Обратим внимание, что для абстракции типа вершины используется статический полиморфизм, когда в случае необходимости производится одновременная компиляция нескольких вариантов одного и того же класса, а для абстракции типа пространства — динамический, при котором методы соответствующих объектов являются виртуальными без жёстко заданного адреса точки входа, точное значение которого вычисляется на основе данных таблицы виртуальных методов.

Ниже приводится перечень имён основных классов библиотеки с кратким описанием назначения:

1. CODEAlgClarkWright — реализация алгоритма Кларка-Райта.
2. CODEAlgLinkSortT — класс для сортировки рёбер по возрастанию стоимости переезда.
3. CODEAlgPrimaT — реализация алгоритма Прима построения минимального остова [4].
4. CODEAlgQuickSortT — реализация алгоритма сортировки Хоара [4].
5. CODEAlgTSP2T — реализация алгоритма 2-опт.
6. CODEAlgTSP3T — реализация алгоритма Лина-Кернигана.
7. CODEAlgTSP6T — реализация алгоритма ветвей и границ для решения ЗК.
8. CODEAlgTSPLocalOptT — реализация алгоритма локальной оптимизации решения ЗК.
9. CODEAlgVRPCapacitatedClusteringT — реализация алгоритма дихотомического деления вершина на группы с учётом грузоподъёмности.
10. CODEAlgVRPCLusteringT — реализация алгоритма дихотомического деления вершина на группы для СЗМТ.

11. CODEAlgVRPPostClusteringT — реализация алгоритма разрезания общего маршрута.
12. CODEBasicLinkT — общий класс для рёбер и дуг.
13. CODECapacitatedAssociationsT — класс для деления вершин на две группы с учётом грузоподъёмности на основе критерия разности.
14. CODECapacityImpossibleException — класс-исключение для представления ошибки о невозможности решить задачу для указанных потребностей в товаре вершин-клиентов.
15. CODEDiffDistGrouperT — реализация деления вершин на две группы на основе критерия разности для СЗМТ.
16. CODEEuclidsSpaceT — класс-пространство для вычисления евклидовых метрик.
17. CODEException — базовый класс для исключений.
18. CODEFNum — класс для выполнения операций сравнения чисел с плавающей точкой.
19. CODEGraphT — класс для хранения информации о графе.
20. CODEGraphT::VertexIterator — класс-итератор для получения списка всех вершин, входящих в граф.
21. CODEGraphT::Iterator — класс-итератор для получения списка всех рёбер, выходящих из некоторой вершины графа.
22. CODEGraphTraverseT — реализация алгоритма поиска в ширину на графе.
23. CODELinkT — ребро между двумя вершинами без учёта направления.
24. CODEMatrix — класс для хранения и вычисления матриц.
25. CODEMatrixSpace — класс-пространство для получения информации о стоимости переезда на основе данных матрицы.
26. CODEMinDistGrouperT — реализация деления вершин на две группы на основе критерия минимума для СЗМТ.
27. CODEOrLinkT — дуга, соединяющая две вершины с учётом направления.

28. CODERandom — реализация алгоритма генерации случайных чисел.
29. CODESimpleLoadPlanner — класс для вычисления уровня загрузки транспортных средств для СЗМТ.
30. CODESpaceT — базовый класс-пространство.
31. CODETableT — двухмерное хранилище объектов произвольного типа. Используется как базовый класс для матриц.
32. CODETanchestraOsman — реализация алгоритма Османа поиска с исключениями.
33. CODETourBuilderT — класс для построения замкнутого маршрута на основе списка рёбер.
34. CODETourT — класс для хранения информации о замкнутом маршруте.
35. CODETSPFloorT — реализация алгоритма вычисления нижней оценки стоимости решения ЗК.
36. CODEVertex2d — вершина в двухмерном пространстве без учёта потребности в товаре.
37. CODEVertex2dQ — вершина в двухмерном пространстве с учётом потребности в товаре.
38. CODEVertexIndex — вершина, представляющая индекс столбца или строки в матрице стоимости переездов, без учёта потребности в товаре.
39. CODEVertexIndexQ — вершина, представляющая индекс столбца или строки в матрице стоимости переездов, с учётом потребности в товаре.

Буква “Т” в конце имени класса означает, что этот класс параметризован.

Замкнутый маршрут представляется при помощи массива входящих в него вершин. Для доступа к ним используется специальный оператор, допускающий неограниченное пространство индексов и выполняющий необходимые вычисления для получения точного номера вершины. Поскольку в некоторых случаях возникает необходимость хранить замкнутые маршруты в виде списка рёбер, был реализован дополнительный алгоритм построения маршрута из них в форму, представленную массивом вершин.

Графы позволяют многократное включение одного и того же ребра, а также способны представить несвязный граф. Внутренние структуры для хранения информации о графе представлены в виде списка смежных вершин, которые помещены в хеш-таблицу. Подобный подход способен увеличить скорость поиска нужной вершины. Особенно это важно для работы с разреженными графами.

Алгоритм просмотра всех рёбер графа поиском в ширину реализован в виде отдельного класса. Этот класс допускает два режима работы: произвольный и с учётом порядка просмотра, когда необходимо задать начальную вершину.

Для представления двумерных хранилищ информации, включая матрицы, создан класс, реализующий поведение таблицы. Для удобства использования он имеет ряд методов для копирования данных в нужной форме из других таблиц, а также данных из указанных массивов. Класс матрицы наследуется от класса таблицы, но дополнен некоторыми операциями, специфичными только для матриц, такими как сложение матриц, перемножение матриц, умножение матриц на число и т. д.

Библиотека существует в двух вариантах: в виде, готовом для внедрения, и в виде, дополненным возможностями, необходимыми для проведения тестирования на случайных данных. К таким возможностям относится подготовка случайных данных, загрузка задач Кристофидеса-Мингоззи-Тосса, а также вывод результатов тестирования в виде XML-кода. Представление данных в виде XML-кода позволяет его лёгкое последующее трансформирование в HTML-файл или некоторый другой формат.

При написании кода использовалось только переносимое подмножество языка C++, что делает его использование возможным на различных операционных системах. Проверена работа библиотеки в средах GNU/Linux и Microsoft Windows.

3.6 Выводы

Процедура построения замкнутых маршрутов при использовании специальных программных пакетов включает в себя следующие этапы:

1. Загрузка данных карты из внешнего файла.
2. Построение транспортного слоя карты.
3. Построение графа транспортной сети.

4. Выбор пунктов обслуживания и депо.
5. Вычисление матрицы стоимости переездов.
6. Запуск алгоритмов.
7. Сохранение результатов.

Большинство алгоритмов не способны работать с несимметричной матрицей стоимости переездов. Предлагается использовать симметричную матрицу, полученную выбором наименьшего значения стоимости среди двух направлений движения. Если отличия между симметричной и несимметричной матрицей не больше, чем в $1 + \varepsilon$ раз, и выполняется правило треугольника, то будем говорить о *почти метрическом пространстве расстояний* с точностью ε . При условии, что есть возможность разворота маршрута, его стоимость при возврате к несимметричной матрице возрастёт не более, чем в $1 + \varepsilon/2$ раз.

Исследованные алгоритмы реализованы в динамически подключаемой библиотеке, интегрированной в геоинформационную систему IndorGIS. Интерфейс пользователя приложения имеет все необходимые элементы для создания транспортного слоя, выбора пунктов обслуживания и запуска алгоритмов для построения кратчайших замкнутых маршрутов.

Библиотека алгоритмов решения ЗМТ создана на языке C++, имеет размер около 8200 строк исходного кода и содержит 70 классов различного назначения. Особое внимание уделено сохранению возможности конструировать на основе созданных классов приложения, способные решать более широкий круг задач, не предусмотренных на этапе проектирования. Библиотека экспортирует две функции, интерфейс которых выглядит следующим образом:

- `int SolveBCVRP(size_t itemCount,
 const double* rawMatrix,
 size_t depoIndex,
 size_t desiredVehicleCount,
 size_t desiredQuality,
 size_t* result);`
- `int SolveVRP(size_t itemCount,
 const double* rawMatrix,
 size_t depoIndex,`

```
size_t desiredQuality,  
size_t vehicleCapacity,  
const size_t* quantities,  
size_t maxResultLen,  
size_t* result);
```

Заключение

На основании материала диссертации можно сделать следующие выводы:

1. В обзоре приведены наиболее известные варианты ЗМТ: ЗМТ в общем виде, ЗМТ с учётом грузоподъёмности, ЗМТ с ограничением количества вершин в маршрутах и ЗМТ для нескольких депо. ЗМТ является NP-трудной задачей. Точный алгоритм на основе метода ветвей и границ не применяется из-за чрезмерно большого времени вычислений. Известные приближённые методы решения ЗМТ делятся на классические эвристики и метаэвристики. К классическим эвристикам относятся конструктивные, двухфазные и улучшающие алгоритмы. Метаэвристики являются общими методами, на основе которых конструируются алгоритмы решения конкретной задачи. Они, как правило, потенциально показывают лучшее качество решений по сравнению с классическими, но их внедрение в программные пакеты затруднено наличием управляющих параметров, а также большей трудоемкостью.
2. Предложена сбалансированная ЗМТ с дополнительным требованием, чтобы количество вершин для каждой пары маршрутов не отличалось бы более, чем на единицу. Количество маршрутов должно быть задано заранее. Рассмотрены алгоритмы заметания и разрезания общего маршрута для сбалансированного вида задачи.
3. На основе процедуры сбалансированного дихотомического деления вершин на группы разработаны эффективные приближённые алгоритмы для решения следующих вариантов ЗМТ: при заданном количестве транспортных средств, ЗМТ с учётом грузоподъёмности и/или с ограничением количества вершин в маршрутах, а также ЗМТ для нескольких депо. Как показал вычислительный эксперимент, разработанные алгоритмы мало уступают по качеству решений одному из лучших метаэвристических алгоритмов для ЗМТ — алгоритму Османа для малых размерностей и превосходят его при больших, обладая при этом существенно меньшей трудоемкостью (порядка $O(n^2)$). Разработана модификация предложенных алгоритмов на основе геометрической информации, обладающая трудоемкостью работы $O(n \log^2 n)$.

4. В виду того, что большинство алгоритмов не способно работать с несимметричной матрицей стоимости переездов, можно эту матрицу симметризовать. Показано, что при отклонении значений симметричной и несимметричной матрицы не более, чем в $1 + \varepsilon$ раз и при выполнении правила треугольника, что соответствует реальным данным, при возврате построенного маршрута к несимметричной матрице его стоимость возрастёт не более, чем в $1 + \varepsilon/2$ раз.
5. Разработанные алгоритмы реализованы в виде отдельной динамически подключаемой библиотеки, которая включена в состав геоинформационной системы IndorGIS и может быть использована в других системах поддержки принятия решений в области транспортной логистики.

Список литературы

1. Алексеев А. О. Экспериментальная оценка эффективности алгоритмов решения задачи -коммивояжеров / А. О. Алексеев, О. Г. Алексеев, О. А. Кулагин // Экономика и математические методы, 1993 № 3. с. 496–502.
2. Большев Л. Н. Таблицы математической статистики / Большев Л. Н., Смирнов Н. В. // М.: Наука. Главная редакция ФМЛ, 1983. 416 с.
3. Гэри М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон // М.: Мир, 1982.
4. Кормен Т. Х. Алгоритмы: построение и анализ / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн // М.: МЦНМО, 1990. 960 с.
5. Меламед И. И. Задача коммивояжера. Приближенные алгоритмы / И. И. Меламед, С. Сергеев, И. Сигал // Автоматика и телемеханика. 1989, № 11.
6. Меламед И. К задаче нескольких коммивояжеров // Межвуз. сб. Вып. 647. М.: МИИТ, 1981.
7. Пожидаев М. С. Исследование алгоритмов приближённого решения сбалансированной задачи k коммивояжеров / Ю. Л. Костюк, М. С. Пожидаев // Информационные технологии: Материалы XLV Международной научно-студенческой конференции (10–12 апреля 2007 г.). — Новосибирск: Изд-во Новосибир. ун-та, 2007. — С. 118–119.
8. Пожидаев М. С. Приближённые алгоритмы решения сбалансированной задачи k коммивояжеров / Ю. Л. Костюк, М. С. Пожидаев // Вестник ТГУ. УВТиИ. — 2008. — № 1(2). — С. 106–112.
9. Пожидаев М. С. Сбалансированная задача k коммивояжеров для нескольких баз / Ю. Л. Костюк, М. С. Пожидаев // Информационные технологии и математическое моделирование (ИТММ — 2008): Материалы VII Всероссийской научно-практической конференции с международным участием (14–15 ноября 2008 г.). — Томск: Изд-во Том. ун-та, 2008. — Ч1. — С. 182–184.
10. Пожидаев М. С. Сбалансированная задача маршрутизации транспортных средств и проблемы практического применения метаэвристических стратегий / Ю. Л. Костюк, М. С. Пожидаев // Информационные технологии и

- математическое моделирование (ИТТМ — 2009): Материалы VIII Всерос. научн.-практ. конф. с межд. участием (13–14 ноября 2009 г.). — Томск: Изд-во Том. ун-та, 2009. — Ч. 2. — С. 233–235.
11. Пожидаев М. С. Сбалансированная эвристика для решения задачи маршрутизации транспорта с учетом грузоподъемности / Ю. Л. Костюк, М. С. Пожидаев // Вестник ТГУ. УВТиИ. — 2010. — № 3.
 12. Пожидаев М. С. Исследования алгоритмов приближённого решения сбалансированной задачи k коммивояжеров // Информационные технологии и математическое моделирование (ИТММ — 2007): Материалы VI Международной научно-практической конференции (9–10 ноября 2007 г.). — Томск: Изд-во Том. ун-та, 2007. — С. 148–149.
 13. Препарата Ф. Вычислительная геометрия: Введение / Ф. Препарата, М. Шеймос // М. : Мир, 1989. — 478 с.
 14. Alfa A.S. A 3-opt based simulated annealing algorithm for vehicle routing problems / A.S. Alfa, S.S. Heragu, M. Chen // Computers & Industrial Engineering. — 1991. — № 21. — P. 635–639.
 15. Aksen D. Open vehicle routing problem with driver nodes and time deadlines / D. Aksen, Z. zyurt, N. Aras // Journal of the Operational Research Society, Volume: 58, Issue: 9 (2006).
 16. Altinkemer K. Parallel savings based heuristic for the delivery problem / K. Altinkemer, B. Gavish // Operations Research. — 1991. — № 39. — P. 456–469.
 17. Barbarosoglu G. A tabu search algorithm for the vehicle routing problem / G. Barbarosoglu, D. Ozgur. // Computers & Operations Research. — 1999. — № 26. — P. 255–270.
 18. Bean J.C. Genetic algorithms and random keys for sequencing and optimization // ORSA Journal on Computing. — 1994. — № 6. — P. 154–160.
 19. Beasley J.E. Route-first cluster-second methods for vehicle routing // Omega. — 1983. — № 11 — P. 403–408.
 20. Bertsimas D.J. A new generation of vehicle routing research: Robust algorithms addressing uncertainty / D.J. Bertsimas, D. Simchi-Levi // Operations Research. — 1996. — № 44. — P. 286–304.

21. Blanton J.L. Multiple vehicle routing with time and capacity constraints using genetic algorithms. In S. Forrest, editor / J.L. Blanton, R.L. Wainwright // Proceedings of the Fifth International Conference on Genetic Algorithms. — Morgan Kaufmann. — San Mateo, CA, 1993. — P. 452–459.
22. Boctor F. F. Optimal solution of column-circular set-partitioning problems / F. F. Boctor and J. Renaud // Working Paper — Faculte des sciences de l'administration, Universite Laval, Canada — 1993. — P. 93–97.
23. Bramel J.B. A location based heuristic for general routing problems / J.B. Bramel, D. Simchi-Levi // Operations Research. — 1995. — № 43. — P. 649–660.
24. Bullnheimer B. Applying the ant system to the vehicle routing problem. In S. Vofi, S. Martello, I.H. Osman, and C. Roucairol, editors / B. Bullnheimer, R.F. Hartl, C. Strauss // Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization. — Kluwer, Boston, 1998a. — P. 109–120.
25. Bullnheimer B. An improved ant system for the vehicle routing problem/ B. Bullnheimer, R.F. Hartl, C. Strauss // Annals of Operations Research, 1998b. — forthcoming.
26. Christofides N. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, C. Sandi, editors/ N. Christofides, A. Mingozzi, P. Toth // Combinatorial Optimization. — Wiley, Chichester, 1979. — P. 315–338.
27. Clarke G. Scheduling of vehicles from a central depot to a number of delivery points / G. Clarke, J.W. Wright // Operations Research. — 1964. — № 12 — P. 568–581.
28. Colorni A. Distributed optimization by ant colonies. In F. Varela and P. Bourguine, editors / A. Colorni, M. Dorigo, V. Maniezzo // Proceedings of the European Conference on Artificial Life. Elsevier. — Amsterdam, 1991.
29. Colorni A. Ant system for job-shop scheduling / A. Colorni, M. Dorigo, V. Maniezzo, M. Trubian // Belgian Journal of Operations Research, Statistics and Computer Science. — 1994. — № 34. — P. 39–53, 1994.
30. Costa D. Ants can colour graphs / D. Costa, A. Hertz // Journal of the Operational Research Society. — 1997. — № 48. — P. 275–305.

31. Desrochers M. A matching based savings algorithm for the vehicle routing problem / M. Desrochers, T.W. Verhoog // Les Cahiers du GERAD G-89-04, Ecole des Hautes Etudes Commerciales de Montreal, 1989.
32. Dorigo M. Ant colony system: A cooperative learning approach for the traveling salesman problem / M. Dorigo, L.M. Gambardella // IEEE Transactions on Evolutionary Computation. — 1997. — № 1. — P. 53–66.
33. Dorigo M. Ant system: Optimization by a colony of cooperating agents / M. Dorigo, V. Maniezzo, A. Coloni // IEEE Transactions on Systems, Man and Cybernetics. — 1996. — № 26. — Part B. — P. 29–41.
34. Dror M. A vehicle routing improvement algorithm. Comparison of a 'Greedy' and a 'Matching' implementation for inventory routing / M. Dror, L. Levy // Computers & Operations Research. — 1986. — № 13. — P. 33–45.
35. Dueck G. New optimization heuristics: The great deluge algorithm and the record-to-record travel // Journal of Computational Physics. — 1993. — № 104. — P. 86–92.
36. Dueck G. Threshold accepting: A general purpose optimization algorithm / G. Dueck, T. Scheurer // Journal of Computational Physics. — 1990. — № 90. — P. 161–175.
37. Durbin R. An analogue approach to the travelling salesman problem using an elastic net method / R. Durbin, D. Willshaw // Nature. — 1987. — № 326. — P. 689–691.
38. Fahrion R. On a principle of chain-exchange for vehicle-routing problems (1-VRP) / R. Fahrion, W. Wrede // Journal of the Operational Research Society. — 1990. — № 41. — P. 821–827.
39. Fisher M.L. A generalized assignment heuristic for vehicle routing / M.L. Fisher, R. Jaikumar // Networks. — 1981. — № 11. — P. 109–124.
40. Foster B. A. An integer programming approach to the vehicle scheduling problem / B. A. Foster, D. M. Ryan // Opl Res. Q. — 1976. — № 27. — P. 307–384.
41. Gambardella L.M. Ant colonies for the Quadratic Assignment Problem / L.M. Gambardella, E.D. Taillard, M. Dorigo // Technical Report IDSIA / 4–97, IDSIA. — Lugano, Switzerland, 1997.

42. Gaskell T.J. Bases for vehicle fleet scheduling // *Operational Research Quarterly*. — 1967. — № 18. — P. 281–295.
43. Gendreau M. Metaheuristics for the vehicle routing problem / M. Gendreau, G. Laporte, J.- Y. Potvin // *Technical Report CRT-963, Centre de Recherche sur les Transports*. — Universit de Montral, jan 1994.
44. Gendreau M. New insertion and postoptimization procedures for the traveling salesman problem / M. Gendreau, A. Hertz, G. Laporte // *Operations Research*. — 1992. — № 40. — P. 1086–1094.
45. Gendreau M. A tabu search heuristic for the vehicle routing problem / M. Gendreau, A. Hertz, G. Laporte // *Management Science*. — 1994. — № 40. — P. 1276–1290.
46. Ghaziri H. Solving routing problems by a self-organizing map. In T. Kohonen, K. Makisara, O. Simula, J. Kangas, editors // *Artificial Neural Networks*. — North-Holland, Amsterdam, 1991. — P. 829–834.
47. Ghaziri H. Algorithmes connexionnistes pour Voptimisation combinatoire : These de doctorat, Ecole Polytechnique / H. Ghaziri. — Federate de Lausanne, Switzerland, 1993.
48. Ghaziri H. Supervision in the self-organizing feature map: Application to the vehicle routing problem. In I.H. Osman and J.R Kelly, editors // *Meta-Heuristics: Theory and Applications*. — Kluwer, Boston, 1996. — P. 651–660.
49. Gillett B.E. A heuristic algorithm for the vehicle dispatch problem / B.E. Gillett, L.R. Miller // *Operations Research*. — 1974. — № 22. — P. 340–349.
50. Glover F. Tabu search: part I // *ORSA J. Comp.* v1. — 1989. — P. 190–206.
51. Glover F. Tabu search: part II // *ORSA J. Comp.* v2. — 1990. — P. 4–32.
52. Glover F. Tabu search methods for optimization // *Feature Issue of European J.Oper. Res.* V.106. - - 1998. — № 2–3.
53. Glover F. Tabu search. / F. Glover, M. Laguna // Boston: Kluwer Acad. Publ., 1997.
54. Goldberg D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

55. Goldberg D.E. Alleles, loci and the traveling salesman problem. In J.J. Grefenstette, editor, D.E. Goldberg and R. Lingle // Proceedings of the First International Conference on Genetic Algorithms. — Lawrence Erlbaum, Hillsdale, NJ, 1985. — P. 154–159.
56. Golden B.L. Implementing vehicle routing algorithms / B.L. Golden, T.L. Magnanti, H.Q. Nguyen // Networks. — 1977. — № 7. — P. 113–148.
57. Haimovich M. Bounds and heuristics for capacitated routing problems / M. Haimovich, A.H.G. Rinnooy Kan // Mathematics of Operations Research. — 1985. — № 10. — P. 527–542.
58. Holland J. H. Adaptation in Natural and Artificial Systems / J.H. Holland. — The University of Michigan Press, Ann Arbor, MI, 1975.
59. Hopfield J.J. Neural computation of decisions in optimization problems / J.J. Hopfield, D.W. Tank // Biological Cybernetics. — 1985. — № 52. — P. 141–152.
60. Jeon G. A vehicle routing problem solved by using a hybrid genetic algorithm / G. Jeon, H. Leep, J. Shim // Computers Industrial Engineering, Volume: 53, Issue: 4 (2007).
61. Johnson D. S. The Traveling Salesman Problem: A Case Study in Local Optimization. Local Search in Combinatorial Optimization / . S. Johnson, L. A. McGeoch // Aarts E. H. L., Lenstra J. K. (eds.). N. Y.: John Willey & Sons, 1995.
62. Johnson D.S. The traveling salesman problem: A case study. In E.H.L. Aarts and J.K. Lenstra, editors, / D.S. Johnson, L.A. McGeoch // Local Search in Combinatorial Optimization. — Wiley, Chichester, 1997. — P. 215–310.
63. Kawamura H. Cooperative search on pheromone communication for vehicle routing problems / H. Kawamura, M. Yamamoto, T. Mitamura, K. Suzuki, A. Ohuchi // IEEE Transactions on Fundamentals, E81-A. — 1998. — P. 1089–1096.
64. Kinderwater G.A.P. Vehicle routing: Handling edge exchanges. In E.H.L. Aarts, J.K. Lenstra, editors / G.A.P. Kinderwater and M.W.P. Savelsbergh // Local Search in Combinatorial Optimization. — Wiley, Chichester, 1997. — P. 337–360.

65. Kohonen T. // *Self-Organization and Associative Memory*. — Springer, Berlin, 1988.
66. Laporte G. *Classical Heuristics for the Vehicle Routing Problem* / G. Laporte, F. Semet // *Les Cahiers du GERAD, G98-54, Group for Research in Decision Analysis*. — Montreal, Canada, 1998.
67. Lin S. Computer solutions of the traveling salesman problem // *Bell System Technical Journal*. — 1965. — № 44. — P. 2245–2269.
68. Lin S. An effective heuristic algorithm for the traveling salesman problem / S. Lin and B. Kernighan // *Operations Research*. — 1973. — № 21. — P. 498–516.
69. Little J. D. C. An algorithm for the traveling salesman problem / J. D. C. Little, K. G. Murty, D. W. Sweeney, C. Karel // *Operations Research*. v11 (1963), pp 972-989.
70. Manolis N. Kritikos The balanced cargo vehicle routing problem with time windows / Manolis N. Kritikos, George Ioannou // *International Journal of Production Economics*, vol. 123, Issue 1, pages 42 – 51. January 2010.
71. Matsuyama Y. Self-organization via competition, cooperation and categorization applied to extended vehicle routing problems // *In Proceedings of the International Joint Conference on Neural Networks*. — Seattle, WA, 1991. — P. 385–390.
72. Oliver I.M. A study of permutation crossover operators on the traveling salesman problem. In J.J. Grefenstette, editor / I.M. Oliver, D.J. Smith, J.R.C Holland // *Proceedings of the Second International Conference on Genetic Algorithms*. — Lawrence Erlbaum, Hillsdale, NJ, 1987. — P. 224–230.
73. Or. I. *Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking*. Ph.D. dissertation. — Northwestern University, Evanston, IL, 1976.
74. Osman I.H. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem // *Annals of Operations Research*. — 1993. — № 41. — P. 421–451.
75. Osman I. H. A comparison of heuristics for the generalised assignment problem // *Working paper, University of Kent, Canterbury, UK, 1990*.

76. Paessens H. The savings algorithm for the vehicle routing problem // European Journal of Operational Research. — 1988. — № 34. — P. 336–344.
77. Pisinger D. A general heuristic for vehicle routing problems / D. Pisinger, S. Ropke // Computers & Operations Research, Volume: 34, Issue: 8 (2007).
78. Potvin J.-Y. The vehicle routing problem with time windows—Part I: Tabu Search / J.-Y. Potvin, T. Kervahut, B.L. Garcia, J.-M. Rousseau // INFORMS Journal on Computing. — 1992. — № 8. — P. 158–164.
79. Potvin J.-Y. Genetic algorithms for the traveling salesman problem // Annals of Operations Research. — 1996. — № 63. — P. 339–370.
80. Potvin J.-Y. A genetic algorithm for vehicle routing with backhauling / J.-Y. Potvin, C. Duhamel, F. Guertin // Applied Intelligence. — 1996. — № 6. — P. 345–355.
81. Potvin J.-Y. The vehicle routing problem with time windows / J.-Y. Potvin and S. Bengio // INFORMS Journal on Computing. — Part II: Genetic search. — 1996. — № 8. — P. 165–172.
82. Renaud J. A fast composite heuristic for the symmetric traveling salesman problem / J. Renaud, F.F. Boctor, G. Laporte // INFORMS Journal on Computing. — 1996. — № 8. — P. 134–143.
83. Rego C. A subpath ejection method for the vehicle routing problem // Management Science. — 1998. — № 44. — P. 1447–1459.
84. Rego C. A parallel tabu search algorithm using ejection chains for the vehicle routing problem In I.H. Osman and J.P. Kelly, editors / C. Rego, C. Roucairol // Meta-Heuristics: Theory and Applications. — Kluwer, Boston, 1996. — P. 661–675.
85. Renaud J. An improved petal heuristic for the vehicle routing problem / J. Renaud, F. F. Bostor, G. Laporte // Journal of Operational Research Society — 1996. — № 47. — P. 329–336.
86. Robuste F. Implementing vehicle routing models/ F. Robuste, C.F. Daganzo, R. Souleyrette II // Transportation Research, 24B. —1990. — P. 263–286.
87. Ryan D. M. Extension of the petal method for vehicle routing / D. M. Ryan, C. Hjorring, F. Glover // J. Opl Res. Soc. — 1993. — № 44. — P. 289–296.

88. Schmitt L.J. An empirical computational study of genetic algorithms to solve order based problems: An emphasis on TSP and VRPTC : Ph.D. Dissertation / L.J. Schmitt; Fogelman College of Business and Economics. — University of Memphis, 1994.
89. Schmitt L.J. An evaluation of a genetic algorithmic approach to the vehicle routing problem // Working paper, Department of Information Technology Management. — Christian Brothers University, Memphis, 1995.
90. Schumann M. Self-organizing maps for vehicle routing problems minimizing an explicit cost function. In F. Fogelman-Soulie, editor / M. Schumann and R. Retzko // Proceedings of the International Conference on Artificial Neural Networks. — Paris, 1995. EC2&Cie. — P. II-401– 406.
91. Stewart W.R. A Lagrangean relaxation heuristic for vehicle routing / W.R. Stewart Jr and B.L. Golden // European Journal of Operational Research. — 1984. — № 15. — P. 84–88.
92. Salhi S. Improvements to vehicle routing heuristics / S. Salhi, G.K. Rand // Journal of the Operational Research Society. — 1987. — № 38. — P. 293–295.
93. Taillard E.D. Parallel iterative search methods for vehicle routing problems // Networks. — 1993. — № 23. — P. 661–673.
94. Thangiah S.R. Vehicle routing with time windows using genetic algorithms // Technical report SRU- CpSc-TR-93-23. — Slippery Rock University, Slippery Rock, PA, 1993.
95. Tang J. Vehicle routing problem with fuzzy time windows / J. Tang, Z. Pan, R. Fung, H. Lau // Fuzzy Sets and Systems, Volume: 160, Issue: 5 (2009).
96. Thangiah S.R. An adaptive clustering method using a geometric shape for vehicle routing problems with time windows. In L.J. Eshelman, editor // Proceedings of the Sixth International Conference on Genetic Algorithm. — Morgan Kaufmann, San Mateo, CA, 1995. — P. 536–543.
97. Thangiah S.R. Algorithms for the vehicle routing problem with time deadlines / S.R. Thangiah, I.H. Osman, R. Vinayagamoorthy, T. Sun // American Journal of Mathematical and Management Sciences. — 1993. — № 13. — P. 323–355.

98. Thompson P.M. Cyclic transfer algorithms for the multivehicle routing and scheduling problems / P.M. Thompson, H.N. Psaraftis // Operations Research. — 1993. — № 41. — P. 935–946.
99. Ulusoy G. The fleet size and mix problem for capacitated arc routing // Eur. J. Opl Res. — 1985. — № 22. — P. 329–337.
100. Van Breedam A. An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints. Ph.D. dissertation. — University of Antwerp, 1994.
101. Vigo D. A heuristic algorithm for the asymmetric capacitated vehicle routing problem // European Journal of Operational Research. — 1996. — № 89. — P. 108–126.
102. Volgenant A. The symmetric traveling salesman problem and edge exchange in minimal 1-trees / A. Volgenant, R. Jonker // European Journal of Operational Research. — 1983. — № 12. — P. 394-403.
103. Whitley D. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In J.D. Schaffer, editor / D. Whitley, T. Starkweather, D. Fuquay // Proceedings of the Third International Conference on Genetic Algorithms. — Morgan Kaufmann, San Mateo, CA, 1989. — P. 133-140.
104. Wren A. Computers in Transport Planning and Operation. — Ian Allan, London, 1971.
105. Wren A. Computer scheduling of vehicles from one or more depots to a number of delivery points / A. Wren and A. Holliday // Operational Research Quarterly. — 1972. — № 23. — P. 333–344.
106. Xu J. A network flow-based tabu search heuristic for the vehicle routing problem / J. Xu, J.P. Kelly // Transportation Science. — 1996. — № 30. — P. 379–393.
107. Yellow P. A computational modification to the savings method of vehicle scheduling // Operational Research Quarterly. — 1970. — № 21. — P. 281–283.

Приложение