Санкт-Петербургский государственный университет
Кафедра математического моделирования энергетических систем

Егорова Анастасия Аркадьевна

Выпускная квалификационная работа бакалавра

Динамический эвристический алгоритм для задачи транспортной маршрутизации с использованием кросс-докинга

Направление 010400
Прикладная математика и информатика

Научный руководитель,
доктор физ.-мат. наук,
профессор
Захаров В. В..

Санкт-Петербург
2016

Содержание

Введение

История задачи транспортной маршрутизации (Vehicle Routing Problem) начинается со статьи [1], где авторы в 1959 году поставили математическую формулировку и решили задачу о поставке бензина на заправочные станции. На сегодняшний день эта задача является одной из самых известных в области комбинаторной оптимизации.

Общий вид задачи VRP выглядит таким образом: m транспортных средств, первоначально находящихся в депо, должны доставить товар n покупателям. Цель задачи - уменьшить затраты на перевозку товара. Решением классической задачи VRP является множество маршрутов, которые начинаются и заканчиваются в депо, и удовлетворяют тому условию, что каждый клиент посещается только один раз. Затраты на перевозку могут быть уменьшены за счет оптимизации пройденной дистанции или использования меньшего количества транспорта.

Но большинство задач VRP на практике выглядят гораздо сложнее, чем классическая, для них существует много ограничений. Эти задачи можно классифицировать следующим образом, как это сделали в статье [2]:

1) Задачи с ограниченной вместимостью автотранспорта (Capacitated VRP – CVRP). Каждое транспортное средство имеет определенную фиксированную грузоподъемность.

2) Задачи с ограничением времени обслуживания (VRP with Time Windows – VRPTW). При решении практических задач часто возникает необходимость учета временных окон. В этой модели каждый клиент имеет индивидуальный временной интервал (окно), в течение которого он может сделать заказ.

3) Задачи с возможностью частичного отказа от товара (VRP with Pick-Ups and Deliveries – VRPPD). При решении данной задачи нужно учитывать, что суммарный вес доставляемых до клиентов и возвращаемых от них в депо товаров не должен превышать допустимую грузоподъемность транспортного средства.

4) Задачи с возможностью возврата товара после завершения обслуживания всех клиентов (VRP with Backhauls – VRPB). Эта задача является

модификацией предыдущей задачи VRPPD. При этом подразумевается, что каждый транспорт сначала развозит все товары, а затем начинает принимать товар от клиентов.

5) Задачи с возможностью обслуживания одного клиента различным транспортом (Split Delivery VRP – SDVRP). В классической задаче VRP все автомобили имеют одинаковую грузоподъемность, но на практике, чаще всего, автопарк состоит из разногабаритных грузовых автомобилей.

6) Задачи с возможностью доставки грузов в течение длительного периода (Periodic VRP – PVRP). В классической задаче VRP обычный период планирования – один день. В задачах с периодической маршрутизацией используется расширенный период планирования на несколько дней.

7) Задачи, в которых для описания поведения клиентов или транспортных средств могут быть использованы случайные переменные (Stochastic VRP – SVRP). В этом случае одна или несколько переменных (клиенты, запросы или время) могут принимать случайное значение.

8) Задачи с возможностью использования транспортных средств из нескольких депо (Multiple Depot VRP – MDVRP). Чтобы справиться с увеличивающимся количеством запросов на доставку товаров грузоперевозящие компании не только увеличивают автопарк, но и создают несколько складов (депо). Как правило, каждый склад имеет свой автопарк и самостоятельно решает задачу маршрутизации автотранспорта. Но в общем случае можно рассматривать вариант, когда для погрузки каждый автомобиль компании может задействовать любой склад, имеющий необходимый товар.

9) Задачи с возможностью дополнительной загрузки транспортного средства по ходу следования (VRP with Satellite Facilities – VRPSF). Классическая задача VRP предполагает, что из-за ограниченной грузоподъемности каждая машина после доставки всего груза, должна вернуться в депо за новой порцией товаров. Однако в некоторых случаях выгоднее произвести дозагрузку на маршруте, без возврата в депо, используя для этого дополнительные транспортные средства.

10) Задачи, в которых используется кросс-докинг. (VRP with Cross-Docking - VRPCD).

В данной работе исследуется задача последнего типа, то есть выполняется условие, что склад, через который проходит товар, является кросс-доком. Кросс-докинг - относительно новая стратегия складирования в логистике. Под ним понимают консолидацию товара, поступающего на склад, для упорядочения его для последующих транспортировок. То есть товар, доставленный от поставщиков, прегруппировывают и отправляют прямо к уходящему транспорту без задержки на складе. Загрузка на складе занимает, как правило, менее 24 часов, иногда менее часа. Таким образом, кросс-докинг не только предоставляет покупателям качественное обслуживание, но и дает существенные преимущества по сравнению с традиционным складированием: снижение затрат на хранение, дополнительное пространство на складе, управление стоимостью и циклами обработки заказов.

Благодаря этим преимуществам, кросс-докинг широко распространен на практике. В том числе, успешное внедрение этой стратегии произвела компания Walmart - один из крупнейших в мире ритейлеров. В системе этой компании товары непрерывно поставляются на кросс-док, где их группируют и затем отправляют в магазины, при этом на складе они не задерживаются. Избегая затрат на хранение, кросс-докинг позволил Walmart поддерживать низкие цены и повысить прибыльность компании. CompUSA - другой важный пользователь кросс-докинга: 70 % товаров компании также проходит через кросс-док.

Задача VRPCD формулируется следующим образом. Товар доставляют от поставщиков несколькими одинаковыми транспортными средствами на кросс-док, там товар консолидируют и немедленно отправляют их ритейлерам теми же транспортными средствами. Примеры маршрута и консолидации товара на кросс-доке представлены на рис. 1 и 2 соответственно.

Одним из важных пунктов является одновременное прибытие транспорта на склад. Если же они прибудут не одновременно, некоторым транспортным средствам придется ждать разгрузки и загрузки. Следовательно, время поставок для всего транспорта должно быть синхронизировано, и товар должен быть правильно консолидирован.При правильной организа-

ции цепи поставок, товар должен доставляться от поставщика ритейлеру непрерывно, без задержек.

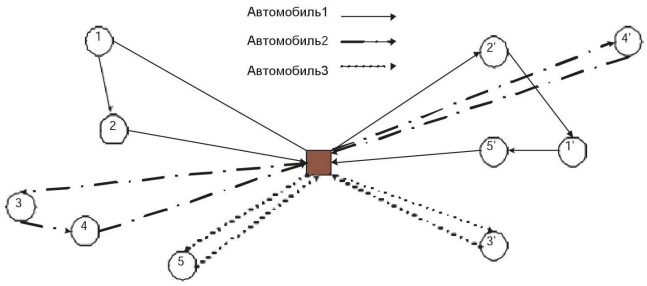

Рис. 1: Пример маршрута.

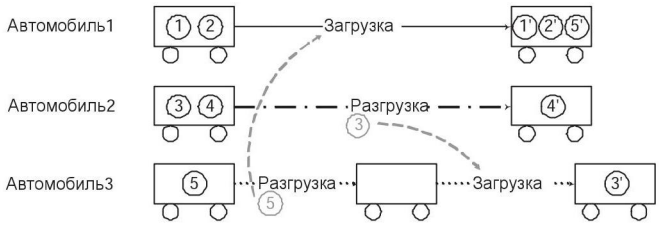

Рис. 2: Пример консолидации товара на кросс-доке.

Целью данной работы является разработка динамического алгоритма, построенного на основе уже существующего эвристического алгоритма, для нахождения оптимальных решений задачи.

Задачи работы:

- Исследование решений задачи транспортной маршрутизации, генерируемых эвристическими методами;
- Разработка динамической адаптации эвристического алгоритма;
- Сравнительная оценка эвристики и динамической адаптации;
- Разработка программного обеспечения, реализующего указанные методы.

Обзор литературы

Большую часть исследований по кросс-докингу занимают его концепция, проектирование и определение местоположения сети. Napolitano в статье [3] рассмотрел различные сферы влияния кросс-дока, такие как производство, распределение, транспортировка, розничная торговля, и выделил все его преимущества. В статье [4] Ratliff, Vate и Zhang рассмотрели проектирование системы кросс-докинга для дистрибьюторской сети, а Apte и Viswanthan [5] описали эту систему как один из самых эффективных способов понижения стоимости транспортировки. Kreng, Chen в статье [6] рассмотрели два подхода к планированию производства и распределения: стратегию традиционного складирования и кросс-докинга, и подтвердили эффективность второй.

Первая версия задачи транспортной маршрутизации при условии кросс-докинга была предложена Lee, Jung, Lee в 2006 [7]. В их рассмотрении все транспортные средства начинают процесс разгрузки товара одновременно после того, как последний автомобиль прибывает на кросс-док. Целью было минимизировать транспортные и операционные издержки, учитывая ограничения транспорного средства по грузоподъемности и временные ограничения поставщиков и ритейлеров. Также они предложили математическую модель данной задачи и эвристический алгоритм Tabu Search.

Wen, Larsen, Clausen, Cordeau и Laporte [8] предложили похожую на предыдущую модель формулировку, однако сняли ограничения на одновременную консолидацию товаров на кросс-доке. Они также предложили Tabu Search и применили его на реальных данных. В результате было найдено решение, уменьшающее функцию затрат на 5 %. Tarantilis [9] обратился к этой же модели и предложил эвристический алгоритм Adaptive Multi-Restart Procedure, связанный с Tabu Search. Алгоритм, предложенный Tarantilis, нашел лучшее решение для 14 из 20 оцениваемых случаев.

Другая версия задачи была предложена Santos, Mateus и da Cunha [10], в которой они не рассматривали временные ограничения поставщиков и ритейлеров. Так же, как и в статье [7], транспортные средства начинают процесс консолидации только после прибытия всех автомобилей. Их основной вклад - новая математическая модель и разработка алгоритма Branch-and-

Price. Для некоторых узлов было получено улучшенное решение.

Позже Santos, Mateus, da Cunha [11] рассмотрели новый тип задачи транспортной маршрутизации - Pickup and Delivery Problem with Cross-Docking. В данной модели рассматриваются два типа маршрутов: поставщик - кросс-док - ритейлер и поставщик - ритейлер. В первом случае транспорт начинает маршрут из кросс-дока, забирает товар у поставщиков, возвращается на кросс-док и затем везет товар ритейлерам. Во втором же случае транспорт едет из кросс-дока к поставщикам, и после последней загрузки едет к ритейлерам, не заезжая при этом на склад. Они также сформулировали математическую модель и применили алгоритм Branch-and-Price, что привело к улучшению результата на 2.5 %.

Также Morais, Mateus, Noronha [12] опираясь на модель, рассмотренной в статье [8], предложили алгоритм Iterated Local Search. Гланое отличие данного алгоритма от предыдущих состоит в том, что здесь рассматриваются только возможные решения, что делает данный алгоритм более эффективным.

Dondo, Méndez и Cerdá в работе [13] сформулировал задачу транспортной маршрутизации при использовании кросс-докинга как задачу целочисленного программирования, в которой в смешанной многоступенчатой сети распределения товары доставляются от поставщика к ритейлеру, используя непосредственно стратегию складирования и/или кросс-докинг.

1 Задача транспортной маршрутизации

1.1 Математическая постановка задачи

Задача транспортной маршрутизации при использовании кросс-докинга может быть определена как ориентированный граф $G =(N, E)$ с множеством узлов $N= P \cup \mathrm{D} \cup\{ 0\}$, где $P =\{ 1, ..., n \}$ - множество поставщиков, $D = \{1', ..., n' \}$ - множество покупателей, 0 - это кросс-док. Пусть $E = \{ (i, j) : i, j \in \mathrm{P}\} \cup \{(i', j') : i', j' \in \mathrm{D}\} \cup \{ (0, \mathrm{i}) : \mathrm{i} \in \mathrm{P}\} \cup \{(0, i') : i' \in \mathrm{D}\}$ - дуги сети. $R = \{ r_1, ..., r_n \}$ - множество запросов, при этом каждый запрос $r \in \mathrm{R}\{ \mathrm{i}_r, i'_r, o_r \}$, где $i_r \in \mathrm{P}, i'_r \in \mathrm{D}$ и $o_r \in \mathrm{N}$ - соответствующие спросы. Также пусть V будет множеством одинаковых транспортных средств с ограниченной вместимостью Q для груза. Каждая дуга $(i, j) \in \mathrm{E}$ связана со временем транспортировки t_{ij} и стоимостью транспортировки c_{ij}. Каждый узел $i \in \mathrm{N}$ связан с временными окнами $[a_i, b_i]$, где a_i и b_i представляют собой минимальное и максимальное время прибытия в этот узел. Задачей VRPCD является нахождение множества S маршрутов для транспортных средств $V^S \subseteq \mathrm{V}$ с целью минимизации затрат на перевозку с учетом спроса в узлах, временных окон и вместимости транспорта. Маршрут транспортного средства делится на два этапа: маршрут Pickup (товар необходимо взять у поставщиков) и маршрут Delivery (доставка товара покупателям). Маршрут Pickup начинается на кросс-доке, проходит через поставщиков и заканчивается на кросс-доке. Маршрут Delivery также начинается на кросс-доке, проходит через покупателей и возвращается на кросс-док. Каждый поставщик и покупатель $i \in \{ \mathrm{P} \cup \mathrm{D} \}$ посещается только один раз.

Транспортные средства могут оставлять и забирать товар с кросс-дока. Однако этот процесс занимает некоторое время. Если транспорт не меняет товар на кросс-доке, время, проведенное там, равно нулю. В случае если транспорту необходимо разгрузить или загрузить товар, время на кросс-доке считается следующим образом. Пусть A будет постоянное время для подготовки транспортного средства, а B - время для разгрузки или загрузки одной единицы товара. Тогда время t_v^u для разгрузки това-

ра, привезенного от поставщиков $P_v \subseteq$ P: $\mathrm{t}_v^u = A + B\sum_{i \in P_v} o_i$. Время для загрузки товара для покупателей $D_v \subseteq$ D: $\mathrm{t}_v^l = A + B\sum_{i \in D_v} o_i$. Транспортное средство может начать движение от кросс-дока до покупателей только после разгрузки товара, привезенного от поставщиков P_v и после того, как весь товар для покупетелей D_v будет загружен. Таким образом, время t_v^{rl}, когда транспортное средство готово к погрузке товара на кросс-доке $t_v^{rl} = max(t_v^a + t_v^u, t_v^{fu})$, где t_v^a - время, когда транспорт v прибывает на кросс-док, а t_v^{fu} - время, когда все транспортные средства, доставляющие товар для дальнейшей транспортировки его транспортом v, заканчивают разгрузку. Тогда время t_v^d, в которое v покидает кросс-док: $t_v^d = t_v^{rl} + t_v^l$, и соответственно время, проведенное на кросс-доке равно $t_v^d - t_v^a$.

Пусть $L = L^p \cup \mathrm{L}^d$, где L^p - множество маршрутов Pickup, L^d - множество маршрутов Delivery. С каждым маршрутом l связана стоимость транспортировки c_q, в нашем случае равная времени транспортировки t_q. Рассмотрим бинарные константы $\lambda_{i,q} \in \{\ 0{,}1\}$ и $\delta_{i'_{,q'}} \in \{\ 0{,}1\}$, которые определяют соответственно посещается ли поставщик $i \in$ P с запросом $r \in$ R в маршруте Pickup $l_q \in \mathrm{L}^p$ и покупатель i' с запросом $r \in$ R в маршруте доставки $l_{q'} \in \mathrm{L}^d$. Также рассмотрим две бинарные переменные $x_q \in \{\ 0,\ 1\}$ и $y_{q'} \in \{\ 0,\ 1\}$, чтобы определить, выбраны ли в качестве пути маршруты l_q и $l_{q'}$. Таким образом, чтобы составить решение задачи VRPCD, необходимо найти маршруты в L^p и L^d, которые минимизируют функционал

$$min \sum_{q \in L^p} c_q x_q + \sum_{q' \in L^d} c_{q' y_{q'}}$$

с учетом ограничений:

$$\sum_{q \in L^p} \lambda_{iq} x_q = 1, \forall \mathrm{i} \in \mathrm{P} \tag{1}$$

$$\sum_{q' \in L^d} \delta_{iq} y_q = 1, \forall i' \in \mathrm{D} \tag{2}$$

$$y_q, x_q \in \{\ 0,\ 1\}, \forall\ \mathrm{l}_q \in \mathrm{L}^p,\ \forall\ \mathrm{l}_{q'} \in \mathrm{L}^d. \tag{3}$$

Функционал показывает, что необходимо минимизировать стоимость решения. Ограничение (1) утверждает, что один маршрут Pickup проходит через поставщика $i \in$ P, а ограничение (2), что один маршрут Delivery

проходит через покупателя $i' \in$ D. Маршруты Pickup и Delivery рассматриваются отдельно друг от друга.

1.2 Алгоритмы решения задачи VRPCD

1. Поиск с запретом (Tabu Search).

   В статье [14] было доказано, что алгоритм поиска с запретом является одним из лучших эвристических алгоритмов, применяемых для задачи VRP. Основная идея этого алгоритма заключается в циклическом переборе решений, в то время как список запретов, сосотоящий из нескольких предшествующих решений, запрещает часть окрестности текущего решения, что позволяет выбраться из локального оптимума.

   Фунция затрат на перевозку в этом алгоритме определяется следующим образом: $f(s) = c(s) + q(s) + d(s) + w(s)$, где $c(s)$ - полная дистанция, пройденная всеми транспортными средствами, $q(s)$ - суммарное превышение вместимости транспорта на всем маршруте, $d(s)$ и $w(s)$ - длительность пути и все нарушения временных окон соответственно.

   Также в этой версии алгоритма этап улучшения решения делится на два фазы: изучение малой окрестности (small neighborhood search) и большой окрестности (large neighborhood search). Во втором случае каждый узел переходит на каждую позицию в маршруте каждого транспортного средства. В первом же случае, вместо исследования всего пространства решений, выбирается множество узлов $N' \subseteq N$, и они перемещаются между определенным набором транспортных средств. Например, узел $i', (i' \in N')$ перемещается к каждому транспорту из M', $(M' \subseteq M)$. Выбираются множества N' и M'_i таким образом: если в текущем решении нарушаются ограничение на вместимость транспортного средства или временные окна какого-либо узла, то все узлы, обслуживаемые этим транспортом, добавляются в множество N'. После того, как будут собраны узлы у всех транспортных средств с нарушениями ограничений, если количество элементов во множестве N' будет меньше заранее заданной константы μ, случайным образом выбираются $\mu - |N'|$ узлов и добавляются в множество N'. В момент

выбора транспортного средства из множества M_i' для обслуживания узла $i' \in N'$, с целью минимизации пройденной транспортом дистанции, выбирается такой транспорт, чтобы узлы, которые он уже обслуживает, находились поблизости от узла i'. Поэтому все узлы $i \in$ N за исключением узла i' сортируются в порядке возрастания дистанции между узлами i и i'. Множество M_i' состоит из всех транспортных средств, которые обслуживают ν узлов, находящихся поблизости от узла i'. Далее, каждый узел из множества N' по очереди включается в ближайшие маршруты транспортных средств.

Стратегия, которую применяют авторы при переключении между small neighborhood search и large neighborhood search заключается в следующем: алгоритм поиска с запретом начинается с исследования малой окрестности и переключается на большую окрестность, если не было улучшений в решении на протяжении последних η итераций. При исследовании большой окрестности, если решение было улучшено за σ итераций, процесс снова переключается на малую окрестность, иначе алгорим останавливается. Ниже представлена полная структура алгоритма поиска с запретом.

Алгоритм Tabu Search

1: $Stop$ = false; $Nb = 2$; $i = 0$; $s* = \emptyset$; f(s*)= ∞;

2: ПОКА $Stop ==$ false

3: ЕСЛИ $Nb == 2$:

4: $SmallNeighbourhoodSearch$;

5: ИНАЧЕ:

6: $LargeNeighbourhoodSearch$;

7: $s =$ решение, найденное при помощи $NeighbourhoodSearch$;

8: ЕСЛИ $f(s) < f(s*)$

9: $s* = s$; $i = 0$; $Nb = 2$;

10: ИНАЧЕ

11: i ++;

12: ЕСЛИ $Nb == 2$ и $i > \eta$

13: $Nb = 1$;

14: ЕСЛИ $Nb == 1$ и $i > \sigma$

15: $Stop$ = true;

16: КОНЕЦ

---

2. Адаптация мультистартового алгоритма (Adaptive Multi-Restart algorithm)

Обычно мультистаротовые методы, аналогично предыдущему алгоритму, состоят из двух этапов: построение решения и улучшение решения. Одна из возможностей избежать зацикливания в локальном оптимуме - это перезапуск поиска из нового решения, если предыдущая окрестность подробно исследована. Ниже представлена полная структура алгоритма мультистартового поиска с запретом.

Алгоритм Adaptive Multi-Restart Tabu Search

1: Начальное решение $(\mu, \nu,$ z$) \rightarrow$ R;

2: ПОКА не выполнены условия остановки:

3: Построение связных решений $(p_c, R) \rightarrow$ s$_g$;

4: $TabuSearch(s_g,\ \nu,$ z, p$_t, R) \rightarrow$ s$_m$;

5: ЕСЛИ $f(s_m) < f(s*)$

6: $s_m \rightarrow$ s*;

7: Временное решение $(s_m) \rightarrow$ R;

8: КОНЕЦ

Сначала формируется начальное решение и записывается в эталонное решение R, после этого начинается сам мультистартовый алгоритм.

Вместо того, чтобы прибегать к помощи случайных процедур, в которых построенное решение не зависит от уже приобретенных за процесс поиска знаний, используется решение R для определения общих элементов. С этой целью используются повторения, построенные на основе эвристического алгоритма, чтобы построить новое временное решение. Для этого запускается алгоритм Tabu Search. Улучшение достигается за счет механизма, который стремится к усилению действий, приводящих к хорошему решению. После того как область изучили, новое локально-оптимальное решение сравнивается с уже существующим, и все повторяется до тех пор, пока не будут достигнуты условия остановки.

Также в алгоритме представлены пять переменных: размер эталонного решения μ, вариационный параметр p_c, и параметры ν, z p_t, которые используются для регулирования процесса локального поиска. В качестве условий остановки выступает процессорное время, которое варьируется в зависимости от количества клиентов.

Данные были взяты те же, что и в статье [8]. В результате мультистартовая адаптация нашла лучшее, по сравнению с алгоритмом Tabu Search, решение в 14 из 20 случаев.

3. Branch-and-Price algorithm

Алгоритм Branch-and-Price является гибридом алгоритмов branch and bound (метод ветвей и границ) и column generation (генерация столбцов). Метод ветвей и границ - это общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, являющийся вариацией полного перебора с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений. Метод состоит из трех этапов: выбора узла, ветвления и нахождения границ. Все узлы рассматриваются по очередности. Процедура ветвления состоит в разбиении множества допустимых значений на подмножества меньших размеров. Если подмножество не содержит оптимального решения, то оно все убирается из рассмотрения. Процедуру можно рекурсивно применять к подобластям. Полученные подмножества образуют дерево, называемое деревом ветвей и границ. Процедура нахождения

границ заключается в поиске верхних и нижних границ для решения задачи на подобласти допустимых значений. [15]

Генерация столбцов - еще один подход к решению задачи линейного программирования. Многие задачи слишком велики для явного выписывания всех переменных. Генерация столбцов поддерживает эту идею путём генерации только тех переменных, которые имеют потенциальную возможность улучшения целевой функции — то есть ищутся только переменные с отрицательной приведенной ценой (при рассмотрении именно задачи минимизации). Задачу можно разделить на две части: основная задача и подзадача. Основная задача является исходной задачей с предположением, что рассматривается только подмножество переменных. Подзадача же — это новая задача, цель которой — поиск новых переменных. Целевой функцией подзадачи является приведённая цена для текущих двойственных переменных, а ограничения порождаются естественными ограничениями на переменные. Процесс работает следующим образом: решаем основную задачу, из решения получаем двойственные переменные для каждого ограничения исходной задачи. Эта информация используется в целевой функции подзадачи. Решаем подзадачу. Если целевая функция подзадачи будет отрицательной, переменная с отрицательной приведённой ценой найдена и эту переменную добавляем в основную задачу и производим очередное решение основной задачи. Новое оптимальное решение основной задачи даст новые двойственные переменные, и процесс повторяется, пока решения подзадачи дают отрицательные значения. Когда решение подзадачи даст положительное значение целевой функции, мы можем заключить, что оптимальное решение основной задачи найдено [16].

Предположим, что основная задача, полученная при помощи ослабления задачи линейного программирования и перестановки маршрутов из P и D, уже решена. Чтобы оценить работу ослабления задачи, которая заключается в замене строгих ограничений (переменные $x_i \in \{ 0,1 \}$) более слабыми ограничениями ($0 \leq x_i \leq 1$), необходимо найти маршруты $p_i \in P$ и $d_i \in D$, которые нарушают ограничения рассматриваемой задачи. Эти маршруты можно найти, решив зада-

чу построения ограниченного начального кратчайшего пути (Resource Constrained Elementary Shortest Path Problems). В то же время, для ее решения используется алгоритм динамического программирования, предложенный в статье [17]. Всякий раз, когда маршрут $p_i \in$ P($\mathrm{d}_i \in$ D соответственно) нарушает ограничения задачи, этот маршрут добавляется в основную задачу и добавляется итерация в алгоритме генерации столбцов. Если решение основной задачи целое, то задача решена, иначе алгоритм запускается заново.

2 Алгоритм Iterated Local Search

2.1 Описание работы алгоритма ILS

ILS представляет собой алгоритм поиска в пространстве локальных оптимальных решений. Он начинается из локального оптимума x, и на каждой итерации выполняет функцию perturbation, которая состоит из случайных шагов в данной окрестности, и находит новый локальный оптимум x'. Если x' удовлетворяет выбранному критерию, процедура повторяется из решения x', иначе алгоритм продолжает работать из x.

Алгоритм Iterated Local Search

1: Начальное решение $\rightarrow$ x;
2: $VND(exchange, relocate, x) \rightarrow$ x, x*;
3: ПОКА не выполнены условия остановки:
4: $Perturbation(x, \rho\ ) \rightarrow x'$;
5: $VND(insertion, swap(1,1), swap(2,1), x') \rightarrow x'$;
6: $LocalSearch(reinsertion, x') \rightarrow x'$;
7: ЕСЛИ выполняется выбранный критерий:
8: $x' \rightarrow$ x
9: ЕСЛИ x лучше, чем $x*$:
10: $x \rightarrow$ x*
11: КОНЕЦ

Начальное решение формируется при помощи жадного алгоритма. Составляется множество запросов $CR \subseteq R$, которые еще не назначены ни одному транспортному средству. Маршруты для каждого транспорта составляются следующим образом: сначала создается пустой узел покупателя для $v \in$ V. Затем, на каждой итерации выбирается запрос $r \in$ CR, и поставщик i_r включается в маршрут Pickup транспорта v, и так далее. Когда заканчивается вместимость данного транспортного средства, алгоритм переходит к следующему. После прибытия на кросс-док, подсчитывается время приезда каждого транспортного средства. Аналогично процессу составления маршрута Pickup, составляется маршрут Delivery для каждого

транспорта. Если количество товара доставки равно количеству товара, забранному у поставщика, время, проведенное на кросс-доке равно нулю, в противном случае учитываются процесс консолидации товара и время прибытия других транспортных средств.

Алгоритм ILS состоит из нескольких алгоритмов локального поиска:

1. Exchange local search. На каждой итерации одному транспортному средству $v_1 \in$ V вместо запроса $r_1 \in$ R, состоящего из поставщика и покупателя, назначается запрос $r_2 \in$ R, и соответственно наоборот: транспорту $v_2 \in$ V вместо r_2 назначается r_1.

2. Relocate local search. В этом алгоритме подсчитывается стоимость транспортного маршрута при условии, что оба маршрута, и Pickup, и Delivery, будет осуществлять транспортное средство v_2, а не v_1.

3. Reinsertion local search. В этом случае происходит замена мест поставщиков (или соответственно покупателей) в маршруте одного транспортного средства. То есть поставщики (покупатели) посещаются транспортом в другом порядке.

4. Insertion local search. В данном алгоритме на каждой итерации поставщик $i \in$ P (соответственно покупатель $i' \in$ D) убирается и маршрута Pickup (Delivery) транспортного средства $v_1 \in$ V и добавляется в маршрут транспортного средства $v_2 \in$ V.

5. Swap(1,1) local search. В этом случае осуществляется замена поставщика $i \in$ P (соответственно покупателя $i' \in$ D) в маршруте Pickup (Delivery соответственно) транспортного средства $v_1 \in$ V с поставщиком $j \in$ D (покупателем $j' \in$ D) в маршруте Pickup (Delivery) транспортного средства $v_2 \in$ V. И наоборот.

6. Swap(2,1) local search. При этом алгоритме происходит замена двух поставщиков $i, j \in$ P (соответственно покупателей $i', j' \in$ D) в маршруте Pickup (Delivery соответственно) транспортного средства $v_1 \in$ V с одним поставщиком $k \in$ P (покупателем $k' \in$ D) в маршруте Pickup (Delivery) транспортного средства $v_2 \in$ V. И наоборот.

Также в алгоритме используется процедура Perturbation, основанная на алгоритмах k-exchange и 2-split. k-exchange, где k=$\rho*$ |V|, ρ - регулируе-

мый параметр, |V| - количество используемого транспорта. Эта процедура строит решение при помощи замены случайно выбранных k поставщиков или покупателей в маршрутах двух транспортных средств. Алгоритм 2-split генерирует новое решение путем разделения одного маршрута текущего решения на два меньших маршрута. Реализацию функции Perturbation можно найти в Приложении 2.

Далее в алгоритме используется процедура Variable Neighborhood Descent (VND), которая является комбинацией набора L алгоритмов локального поиска и в результате дает решение, которое является оптимальным для всех этих алгоритмов. Задана перестановка $\pi = \{\pi_1, ...,$
$\pi_{|L|}\}$, которая представляет собой порядок действия алгоритмов локального поиска, и начальное решение x. Если решение, найденное алогоритмом π_i, не улучшило предыдущее решение, в процедуре происходит переход к алгоритму π_{i+1}, в противном случае переменной x присваивается новое значение, и процедура начинается с алгоритма π_1. Процедура VND останавливается, когда алгоритм $\pi_{|L|}$ не может найти лучшего решения. Реализацию функции VND можно найти в Приложении 3.

В качестве условия остановки берется время, равное 1200s, а в качестве выбранного критерия - стоимость перевозки. Если x' не больше, чем α $f(x)$, где $f(x)$ - стоимость перевозки товара по маршруту x, а α-регулируемый параметр, то решение x заменяется на решение x'.

2.2 Применение алгоритма ILS

Для проведения эксперимента были рассмотрены 5 тестовых задач разных размерностей. Данные находятся в свободном доступе на сайте http://www.homepages.dcc.ufmg.br/ fsantos/instances/. На рис. 3 представлен формат данных для задачи.

Vehicle number обозначает максимальное количество эксплуатируемых транспортных средств, capacity - вместимость каждого транспортного средства. Cust. No. показывает номер узла, нулевой узел - кросс-док, узлы 1-10 - узлы поставщиков, 11-20 - узлы покупателей. Также каждому узлу соответствуют координаты X и Y, расстояние между вершинами вычисляется по евклидовой метрике. Кроме этого, заданы спрос и временные

```
VEHICLE
NUMBER     CAPACITY
  25         33

CUSTOMER
CUST NO.   XCOORD.     YCOORD.     DEMAND   READY TIME   DUE DATE
TIME

    0      677.908     6150.22       0         360         1320
    1      692.484     6114.23       9        419          539
    2      684.846     6102.92       9        366          486
    3      689.324     6103.72       5        362          482
    4      684.314     6087.47       9        455          575
    5      712.94      6191.16      14        377          497
    6      709.708     6191.23       5        415          535
    7      710.492     6187.67       6        388          508
    8      716.807     6192.5       13        466          586
    9      718.45      6179.79      13        390          510
   10      719.311     6181.88       12        432          552
   11      526.379     6098.38       9        741          861
   12      523.451     6254.88       9        812          932
   13      593.371     6136.27       5        673          793
   14      565.869     6083.52       9        797          917
   15      525.886     6101.09       14        781          901
   16      523.383     6254.65       5        789          909
   17      573.018     6225.46       6        785          905
   18      530.562     6153.37       13        826          946
   19      529.291     6148.52       13        717          837
   20      533.717     6170.45       12        973          1093
```

Рис. 3: Фрагмент файла для тестовой задачи.

окна для каждого узла.

В таблице 1 представлены решения, найденные при помощи жадного алгоритма, то есть начальное решение.

η	100	150	200	250	300
$f(x)$	12186	14667	24414	25750	31157

Табл. 1: Результаты реализации жадного алгоритма.

В рассматриваемой задаче существуют три параметра: α, влияющий на стоимость перевозки, ρ - процентная доля в процедуре perturbation, и η - количество поставщиков и покупателей.

Эти параметры принимают значения: $\alpha \in \{$ 2%, 5%, 10%, 15%$\}, \rho \in \{$ 5%, 10%, 20%, 50%$\}, \eta \in \{$ 100, 150, 200, 250, 300$\}$.

Реализацию алгоритма Iterated Local Search на языке C++ можно найти в Приложении 4.

η						
$\alpha, \%$	$\rho, \%$	100	150	200	250	300
2	5	9525	11075	17963	19750	27350
	10	9514	10973	18100	19854	27425
	20	9576	11372	18256	19907	27502
	50	9682	11413	18448	20009	27606
5	5	9598	11180	18324	20014	27408
	10	9577	11176	18205	20098	27476
	20	9636	11484	18436	20145	27543
	50	9715	11523	18687	20204	27658
10	5	9613	11203	18407	20284	27495
	10	9602	11195	18385	20379	27517
	20	9665	11468	18523	20468	27564
	50	9747	11539	18698	20504	27683
15	5	9635	11249	18452	20369	27534
	10	9626	11214	18412	20426	27579
	20	9684	11481	18586	20494	27648
	50	9758	11562	18721	20573	27716

Табл. 2: Результаты реализации алгоритма ILS

Было проведено по 10 экспериментов при различных параметрах α, ρ и разном количестве узлов. В таблице 2 представлены наилучшие решения - время поездки всех транспортных средств для каждого случая. Результаты, приведенные в этой таблице, показывают не только, что алгоритм ILS эффективнее, чем жадный алгоритм для всех рассмотренных случаев, но также, что наименьшие значения целевой функции достигаются при $\alpha = 2\%, \rho = 10\%$ в случае, если узлов менее 150, и при $\alpha = 2\%$, $\rho = 5\%$, если узлов более 150. Улучшение решения при помощи эвристического алгоритма находится в пределах 12,2-26,3%.

3 Динамическая адаптация алгоритма ILS

3.1 Временная состоятельность

Эвристические алгоритмы дают приемлемое решение задачи в большинстве случаев, но не гарантируют его оптимальность. Это означает, что нарушается принцип оптимальности Беллмана, который утверждает, что «оптимальное решение обладает таким свойством, что независимо от начальных данных и начальной точки, оставшееся решение будет оптимальным, с учетом реализации первой точки» [18]. То есть оптимальное решение сохраняет оптимальность на любых подзадачах. Под временной состоятельностью понимают именно это свойство. С другой стороны, подзадачи имеют меньшую размерность, то есть меньшее пространство поиска, что повышает вероятность нахождения оптимального результата за меньшее время. Таким образом, можно найти лучшее решение, оставаясь в рамках эвристического алгоритма.

Так как решение задач маршрутизации можно представить в виде последовательности точек, которые посещает транспортное средство, то к ним применима оценка временной состоятельности.

Введем следующие обозначения:

пусть P - множество тестовых примеров для задачи, $p \in$ P - соответственно один пример из этого множества, $s(p)$ - множество решений, полученное при помощи эвристического алгоритма для задачи. Каждое такое решение представляет собой порядок обхода городов для каждого транспортного средства в течение T периодов, эта величина постоянна. Исходное решение $s(p)$ разделим на части, соответствующие каждому из периодов t = 0, 1,..., T-1. Количество узлов, пройденных в течение первых t периодов, вычисляется по формуле $n(t, s(p)) = \frac{n_0 t}{T}$, где n_0 - количество узлов в тестовой задаче. $s^+(t,p)$ - часть маршрута $s(p)$, соответствующая порядку обхода узлов после периода t, а $s^-(t,p)$ - до периода t соответственно. Тогда каждое решение можно представить в виде объединения двух частей $s(p) = s^+(t,p) \cup \mathrm{s}^-(t,p)$.

Рассмотрим текущую задачу $p(s^-(t,p))$, которая отличается от начальной тестовой тем, что сокращено множество узлов, которые уже входят в часть маршрута $s^-(t,p)$. Через $s(p(s^-(t,p)))$ обозначим решение

текущей задачи, найденное при помощи эвристического алгоритма.

Определение. Решение $s(p)$ называется состоятельным по времени, если для каждого t = 0, 1,..., T-1 верно следующее неравенство

$$f(s^{+}(t,p)) \leq f(s(p(s^{-}(t,p)))), \tag{4}$$

где f - целевая функция рассматриваемой задачи.

Для каждого решения $s(p)$ проведем M вычислительных экспериментов, которые состоят в проверке решения на состоятельность по времени. Через $b(s(p),t)$ обозначим число экпериментов, для которых нарушается это свойство после периода t. Если бы решение было оптимальным, то, согласно принципу оптимальности Беллмана, выполнялось бы равенство $\sum_{k=1}^{T-1} b(s(p),t) = 0$.

Определение. Экспериментальным уровнем временной состоятельности алгоритма называется величина, определяемая по следующей формуле

$$con = 1 - \frac{1}{M|P|} \sum_{p \in P} \frac{1}{|S(p)|} \sum_{t}^{T-1} b(s(p),t), \tag{5}$$

где $S(p)$ - множество решений, сгенерированных эвристическим алгоритмом для задачи p.

Заметим, что $0 \leq$ con≤ 1. Чем ближе значение оценки временной состоятельности к 0, тем чаще решение, полученное эвристическим алгоритмом, теряет свойство оптимальности в ходе своей реализации.

3.2 Описание работы динамической адаптации

Метод динамической адаптации алгоритмов для задачи маршрутизации и управления запасами подробно был рассмотрен в статье [19]. Используя его идею, адаптируем его для рассматриваемой в этой работе задачи.

Динамическую адаптацию можно описать следующим образом: на начальном этапе для каждого узла при помощи алгоритма Iterated Local Search генерируется N решений. Из них выбирается то, которое обладает наименьшим значением целевой функции. Каждый маршрут делится на T периодов, после каждого периода задачи расссматривается новый этап, но уже с меньшим количеством узлов для каждого транспорта. На каждом этапе снова запускается эвристический алгоритм. Если на новом этапе уда-

ется найти лучшее решение, то маршрут меняется. Улучшения эвристического алгоритма таким способом возможны из-за уменьшения количества рассматриваемых узлов на каждом этапе.

Общая схема динамической адаптации приведена ниже.

Динамическая адаптация эвристического алгоритма

1: Генерируется n решений эвристического алгоритма, определяется лучшее $\rightarrow$ p;
2: ПОКА $t < T$
3: генерируется n решений
4: выбирается решение с наименьшим значением целевой функции на p';
5: ЕСЛИ $f(p') \leq$ f(p);
6: маршрут следует изменить на p';
7: КОНЕЦ

3.3 Применение динамической адаптации

В среднем, маршрут для каждого транспортного средства, сгенерированный при помощи алгоритма Iterated Local Search, состоит из 3-5 узлов. Следовательно для проведения эксперимента возьмем параметр T = 3. В качестве всего множества задач P рассмотрим одну задачу с различным количеством узлов, т. е. $|P| = 5$. Для каждой тестовой задачи сгенерируем по 10 решений, количество тестов M для одного решения $s(p)$ равно 5.

По результатам экспериментов был найден средний экспериментальный уровень временной состоятельности для алгоритма Iterated Local Search:

$$\text{conILS}=0.385,$$

что означает, что чуть больше трети решений, сгенерированных на начальном этапе, сохраняют свойство оптимальности в процессе своей реализации, т. е. существуют другие маршруты, которые можно получить при помощи динамической адаптации, значение целевой функции которых

будет меньше по сравнению с начальным решением. Реализацию линамической адаптации алгоритма на языке C++ можно найти в Приложении 5.

Ниже представлены результаты динамической адаптации алгоритма Iterated Local Search для задач с различным количеством узлов при T=3.

T	η				
	100	150	200	250	300
3	9514	10908	17640	19118	26146

Табл. 3: Результаты реализации динамической адаптации.

При анализе данных из таблиц 2 и 3 можно заметить, что в большей части случаев динамичексий алгоритм показал лучшие значения. В примерах малой размерности полученное эвристическим алгоритмом решение уже является оптимальным или близким к оптимальному, поэтому получить улучшение не удалось. На примерах большей размерности алгоритм показывает растущую эффективность, так как при увеличении размерности эффективность алгоритма ILS уменьшается, появляется возможность дальнейшей оптимизации алгоритма. Улучшение решения при помощи динамической адаптации по сравнению с эристическим алгоритмом находится в пределах 0-4,4%.

Заключение

В данной работе были рассмотрены несколько видов задачи транспортной маршрутизации и подробно исследован один из этих видов - задача транспортной маршрутизации при использовании кросс-докинга.

Были изучены различные эвристические подходы к решению данного типа задач. Среди них был выбран алгоритм Iterated Local Search, который был реализован на языке C++ и рассмотрен на тестовых данных. Оценка результатов эксперимента выявила оптимальные значения параметров для различного количества данных и показала преимущество использования эвристики перед жадным алгоритмом. Значения, полученные при помощи алгоритма ILS, превосходят полученнные в среднем на 20%. Также была произведена оценка уровня временной состоятельности алгоритма, среднее значение данной величины равно 0,385, что означает, что эвристика не дает оптимального решения.

Также был разработан метод динамической адаптации алгоритма ILS и реализован в программном коде. Значение решения, сгенерированного этим алгоритмом, получилось меньше, чем алгоритмом ILS. Улучшение значения решения в экспериментах для задачи с различным количеством узлов принимает до 4.4%.Результаты показали, что использование на практике алгоритма динамической адаптации позволяет генерировать маршруты меньшей длины, и, соответственно, меньшие по времени.

Список литературы

[1] Dantzig G. B., Ramser J. H., The Truck Dispatching Problem, 1959

[2] Гладков Л. А., Гладкова Н. В., Особенности и новые подходы к решению динамических транспортных задач с ограничением по времени // Известия ЮФУ. Технические науки, 2014

[3] Napolitano M., Making the Move to Cross Docking: a Practical Guide to Planning, Designing, and Implementing a Cross Dock Operation // WERC Publisher, 2000.

[4] Ratliff H. D., Vate J. V., Zhang M., Network design for load-driven cross-docking systems // Technical Report, The Logistics Institute, Georgia Institute of Technology, 1999.

[5] Apte U. M., Viswanthan S., Effective cross docking for improving distribution efficiencies // Int. J. Logist.-Res., 2000

[6] Kreng V. B., Chen F. T., The benefits of a cross-docking delivery strategy: a supply chain collaboration approach // Production Planning and Control, 2008

[7] Lee Y. H., Jung J. W., Lee K. M. Vehicle routing scheduling for crossdocking in supply chain // Computers & Industrial Engineering, 2006

[8] Wen M., Larsen J., Clausen J., Cordeau J.-F., Laporte G., Vehicle routing with cross-docking // Journal of The Operational Research Society, 2009

[9] Tarantilis C. D. Adaptive multi-restart Tabu Search algorithm for the vehicle routing problem with cross-docking, 2012

[10] Santos F. A., Mateus G. R., Salles da Cunha A. A Branch-and-price algorithm for a Vehicle Routing Problem with Cross-Docking // Electronic Notes in Discrete Mathematics, 2011

[11] Santos F. A., Mateus G. R., Salles da Cunha A. The Pickup and Delivery Problem with Cross-Docking // Computers & Operations Research, 2012

[12] Morais V., Mateus G., Noronha T. Iterated local search heuristics for the Vehicle Routing Problem with Cross-Docking // Expert Systems with Applications, 2014

[13] Dondo R., Méndez C. A., Cerdá J., The multi-echelon vehicle routing problem with cross docking in supply chain management // Computers and Chemical Engineering, 2011

[14] Cordeau J.-F., Gendreau, M., Laporte, G., Potvin, J.-Y., Semet, F., A guide to vehicle routing heuristics // Journal of the Operational Research Society, 2002.

[15] Clausen J., Branch and Bound Algorithms—Principles and Examples. Technical report, 1999

[16] Wikipedia. Генерация столбцов. https://ru.wikipedia.org/wiki/Генерация_столбцов.

[17] Feillet, D., and Dejax, P., and Gendreau, M., and Gueguen, C., An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems // Networks 44, 2004.

[18] Беллман Р. Динамическое программирование. М.: Изд-во иностранной литературы, 1960. 400 с

[19] Shirokikh, V. A., Zakharov, V. V. Dynamic Adaptive Large Neighborhood Search for Inventory Routing Problem. // Advances in Intelligent Systems and Computing, Vol. 359, 2015.

Приложения

Приложение 1: код функции VND(exchange, swap)

```
float VNDinit()
{
	float AnsExchange=Exchange();
	if (AnsExchange<=Ans)
	{
		Solution.Route; // сохранение маршрута
		Ans=AnsExchange;
	}
	float AnsRelocate=Relocate();
	if (AnsRelocate<=Ans)
	{
		Solution.Route;
		Ans=AnsRelocate;
	}
return (Ans);
}
```

Приложение 2: код функции Perturbation

```
float Perturbation()
{
	srand((unsigned)time(NULL));
	int r=rand() % 200;
	int rand=r%2;
	if (rand==0)
	{
		float ansPert=kExchange();
		if (ansPert<=Ans)
		{
			Solution.Route;
			Ans=ansPert;
		}
	}
	else
	{
		float ansPert=Split();
		if (ansPert<=Ans)
		{
			Solution.Route;
			Ans=ansPert;
		}
	}
return (Ans);
}
```

Приложение 3: код функции VND

```
float VND()
{
	float AnsInsertion=Insertion(); //для маршрута Pickup
	loop:
	if (AnsInsertion<=Ans)
	{
		Solution.Route;
		Ans=AnsInsertion;
		goto loop;
	}
	AnsInsertion=Insertion(); //для маршрута Delivery
	if (AnsInsertion<=Ans)
	{
		Solution.Route;
		Ans=AnsInsertion;
		goto loop;
	}
	float AnsSwap11=Swap11();
	if (AnsSwap11<=Ans)
	{
		Solution.Route;
		Ans=AnsSwap11;
		goto loop;
	}
	float AnsSwap21=Swap21()
	{
		Solution.Route;
		Ans=AnsSwap21;
		goto loop;
	}
return(Ans);
}
```

Приложение 4: код алгоритма ILS

```
float ILS()
{
        float alpha, rho, funct;
        clock_t delay=1200;
        clock_t start=clock();
        float AnsILS=VNDinit();
        while (clock()-start<delay)
        {
                AnsILS=Perturbation();
                AnsILS=VND();
                bool a=true;
                AnsILS=Reinsertion(); // для маршрута PIckup
                bool a=false;
                AnsILS=Reinsertion(); // для маршрута Delivery
                funct=Solution(); // подсчитывает время перевозки
                                        на маршруте Pickup
                funct+=SolutionWH(); // подсчитывает время на
                                                кросс-доке
                funct+=Solution(); // время перевозки на маршруте Delivery
                if (AnsILS<=alpha*funct)
                {         Solution.Route;
                        float ansx=Solution();
                        ansx+=SolutionWH();
                        ansx+=Solution();
                        float ansz=Solution();
                        ansz+=SolutionWH();
                        ansz+=Solution();
                        if (ansx>ansz)
                             Solution.Route;
                }
        }
        return (AnsILS);
}
```

Приложение 5: код динамической адаптации алгоритма ILS

```
float DILS(AnsILS)
{
        int T=3;
        float AnsDILS, ans, ansTemp=0;
        v1=rand() % nv;
        v2=rand() % nv;
        for (int i=0; i<nv; i++)
        {
                if (RouteN[i]>maxVehNum)
                        maxVehNum=RouteN[i];
        }
        for (int t=1; t<T; t++)
        {
                beg=t-1;
                fin=t;
                ans=ansTemp+ILS(beg);
                ansTemp+=ILS(beg, fin);
                if (ans<AnsILS)
                {
                        AnsDILS=ans;
                        Solution.Route;
                }
        }
        return(AnsDILS);
}
```