Обзор методов решения задачи о приёмке и доставке с временными ограничениями. Часть II: эвристический подход

Н.В. Царьков, Д.Ю. Голембиовский

***Аннотация* — В данной статье рассматривается задача о приемке и доставке с временными ограничениями *(Pickup and Delivery Problem with Time Windows, PDTPW)* и для её решения предлагается модификация локального поиска – адаптивный поиск в больших окрестностях *(Adaptive Large Neighbourhood Search, ALNS)*, заключающийся в применении некоторых операторов разрушения и восстановления решения. Алгоритм дополнен механизмом запретов *(tabu)*, что позволяет избегать зацикливания при поиске и способствует более широкому исследованию пространства решений. Предлагаемый подход демонстрирует хороший баланс между точностью решений и временем вычислений, и позволяет легко адаптироваться под различные стохастические или динамические расширения задачи PDPTW. Эффективность метода подтверждена на тестовых наборах Li & Lim и Sartori.**

***Ключевые слова* — VRP, PDPTW, ILP, ALNS, local search, локальный поиск, эвристики, метаэвристики, tabu search, маршрутизация, транспортные задачи.**

I. Введение

Данная статья является продолжением статьи [1] о методах решения задачи о приёмке и доставке с временными ограничениями *(Pickup and Delvery Problem with Time Windows, PDPTW)*. В такой задаче нам дано несколько запросов на доставку, состоящих из пункта приемки и пункта доставки, парк транспортных средств, находящийся в депо; известна стоимость и время проезда между парами точек, а также от каждой точки до депо и обратно. Необходимо построить маршруты для транспортных средств, начинающиеся и заканчивающиеся в депо, такие, чтобы каждый запрос выполняло только одно транспортное средство, т.е. посещало и точку приемки, и точку доставки этого запроса, при этом соблюдало все временные ограничения и ограничения на вместимость транспортных средств, и чтобы общая суммарная стоимость проезда по маршрутам была минимальной.

Задача PDPTW относится к классу NP-сложных задач, так как является одним из видов задач маршрутизации транспортных средств *(Vehicle Routing Problem, VRP)* [2], а поэтому при их решении необходимо помнить, что вычислительная сложность резко возрастает при увеличении размерности задачи. При больших размерностях использование точных методов, например, метода ветвей и отсечений *(branch and cut)*, становится неэффективным, так как время на поиск такого решения становится очень огромным, что зачастую не вписывается в требования бизнеса. В такой ситуации использование эвристических методов оправдывается тем, что они позволяют за разумное время найти приемлемые решения, пусть и не оптимальные.

Цель данной статьи – предложить для решения задачи PDPTW алгоритм, основанный на адаптивном поиске в больших окрестностях *(Adaptive Large Neighbourhood Search, ALNS)* и дополненный гибким механизмов запретов. Данный подход также можно использовать и на различных расширениях задачи. Работа сопровождается численными экспериментами на стандартных наборах данных.

Статья организована следующим образом: в 2-м разделе предлагается начальный обзор литературы. В 3-м разделе дается основная терминология и математическая формулировка задачи о приемке и доставке с временными ограничениями. В 4-м и 5-м разделе подробно описывается алгоритм решения задачи. В 6-м разделе приводится численное исследование эффективности алгоритма.

II. Обзор эвристических методов решения задачи

Задача, рассматриваемая в данной статье, имеет некоторые общие черты с транспортными задачами, ранее изученными в литературе и названными выше. Их практическая значимость проявляется в различных областях: от электронной коммерции до медицинской логистики [12].

Классические эвристики, в том числе методы построения маршрутов, предложенные Solomon [3], до сих пор широко используются для построения выполнимых решений. Более поздние работы, такие как Lau и др. [4], рассматривают локальный поиск в качестве алгоритма для последовательного улучшения начального решения.

Среди метаэвристических подходов важное место занимает адаптивный поиск в больших окрестностях *(Adaptive Large Neighborhood Search, ALNS)*, предложенный Ropke и Pisinger [5] [6] для задачи VRP и

Статья получена 13 июня 2025.
Царьков Никита Владимирович, ВМК, МГУ имени М.В. Ломоносова (e-mail: tsarkov90@gmail.com).

Голембиовский Дмитрий Юрьевич, д.т.н., профессор, ВМК, МГУ имени М.В. Ломоносова (email: golemb@cs.msu.ru).

расширяющий локальный поиск, используя набор операторов разрушения и восстановления, динамически адаптирующийся к различным типам задач и масштабам.

Поиск с запретами *(Tabu Search)* [11] [14] и метод имитации отжига *(Simulated Annealing)* до сих пор остаются популярными методами решения задач большой размерности. Li и Lim [7], Nanry и Barnes [8] предложили метаэвристики, объединяющие эти два подхода.

Так же встречается применение различных био-вдохновлённых методов, таких как алгоритм муравьиной колонии *(Ant Colony Optimization)* [9], и генетических алгоритмов (GA) [10].

В последнее время имеется тенденция к интеграции методов машинного обучения с эвристическими подходами. Работы Fang и др. [17] и Rabecq с Chevrier [18] демонстрируют потенциал нейросетей, в частности механизмов внимания *(attention)*, и глубокого обучения с подкреплением *(Deep Reinforcement Learning)*. Однако у нейросетевых методов имеются недостатки: они требуют много вычислительных ресурсов, времени и данных, что скорее всего и является причиной их пока что малой популярности при решении транспортных задач.

Таким образом, современное развитие эвристических методов для решения задачи PDPTW представляет собой переход от традиционных алгоритмов к адаптивным и гибридным системам.

III. Задача о приёмке и доставке с временными ограничениями (Pickup and Delivery Problem with Time Windows)

Напомним основную терминологию задачи:

Даётся K – парк идентичных транспортных средств и Q_{max} – их вместимость (в мерах объема). $P = \{1, \dots, n\}$ и $D = \{n+1, \dots, 2n\}$ – наборы пунктов, откуда нужно забрать и куда нужно доставить груз соответственно ($|P| = |D| = n$). Определим $N = P \cup D$ как множество клиентских пунктов, а пункты с индексами 0 и $2n+1$ как депо, откуда машины начинают и где заканчивают свой маршрут. Каждый пункт $i \in N$ имеет своё время на обслуживание $s_i \geq 0$ и интервал обслуживания $TW_i = [a_i, b_i]$. Отметим, что машина должна приезжать в пункт i не позднее времени b_i, а если она приедет в пункт i раньше времени a_i, то вынуждена будет ждать до времени a_i. Множество всех пунктов, используемых в задаче, определим как $V = N \cup \{0,\ 2n+1\}$.

$R = \{r_1, \dots, r_n\}$ – это набор запросов на доставку, где каждый запрос $r \in R$ представлен пунктом $p_r \in P$, откуда нужно забрать груз размером $q_r \geq 0$ и пунктом $d_r \in D$, куда необходимо доставить этот груз. Обозначим за $r(i)$ – запрос, связанный с клиентским пунктом $i \in N$.

Вводится ориентированный граф $G = (V, A)$, где A – это множество дуг $A = V \times V$, за исключением тех, которые ведут к невыполнимым решениям: мы исключаем дугу (i, j) из пункта i в пункт j, если $b_j < a_i + s_i + t_{i,j}$, где $t_{i,j}$ и $c_{i,j}$ – время и расходы на преодоление расстояния $\rho_{i,j}$ между пунктами i и j.

Задача PDPTW заключается в построении таких маршрутов для транспортных средств, чтобы все запросы на доставку были выполнены с минимальными расходами с учётом всех ограничений.

Обратим внимание, что существуют различные варианты задачи PDTPW, например, с запросами, обслуживающими несколько пунктов приемки и доставки. Способы перехода к классической постановке PDPTW, обсуждаются в 1-й части статьи [1].

На рис. 1–3 представлен графический пример задачи о приемке и доставке. На рис.1 изображено расположение пунктов: зелёными точками обозначены пункты приёмки, а синими– пункты доставки, а пунктирными линиями между ними – запросы: пары пунктов приёмки и доставки (например, пункты 8 и 19 образуют запрос). На рис. 2 (слева) на временной оси расположены временные окна для каждого клиентского пункта. На рис 3 (справа) показаны размеры грузов для каждого запроса

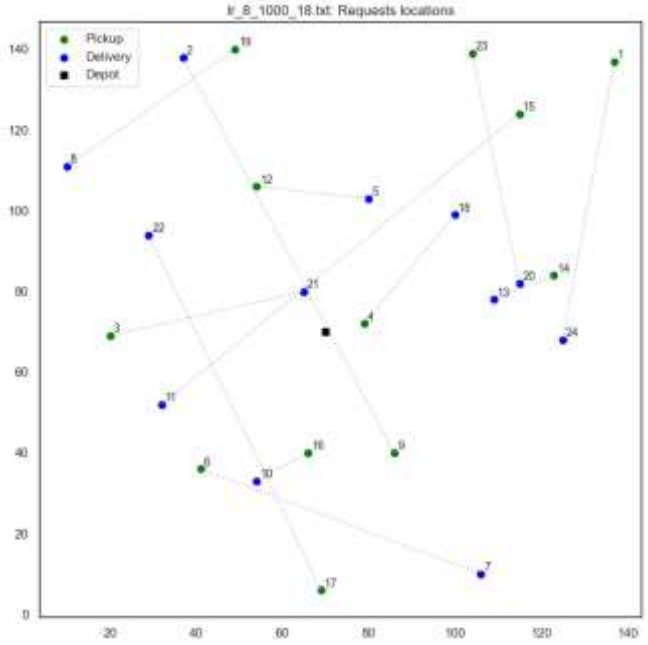

Рис. 1. Пример задачи о приёмке и доставке. На карте точками обозначены пункты (черным квадратом – депо, зелеными точками – пункты приёмки, синими – пункты доставки), пунктирными линиями обозначены запросы: пары пунктов приемки и доставки

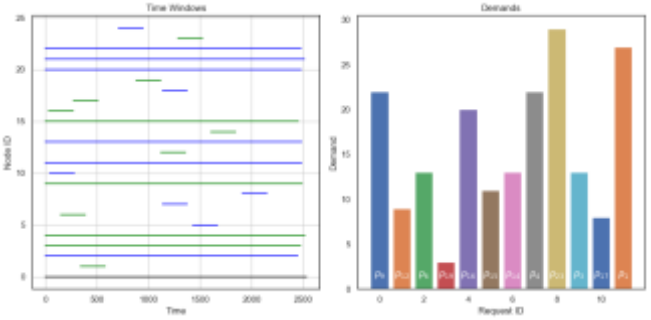

Рис. 2–3. Временные окна и размеры грузов для примера задачи PDPTW с рис. 1. Слева изображены временные окна для клиентских пунктов (зеленые линии – для пунктов приёмки, синие – для пунктов доставки). Справа изображены размеры грузов для каждого запроса, внутри столбцов вписаны пункты приёмки для каждого запроса

На рис. 4 представлено решение задачи, изображённой на рис. 1–3. На нём замкнутой последовательностью стрелок одинакового цвета изображается маршрут прохождения пунктов для определённой машины (например, последовательность «депо → 15(р) → 1(р) →

24(d) → 9(p) → 19(p) → 2(d) → 8(d) → 3(p) → 11(d) → 21(d) → депо» является одним из маршрутов в решении).

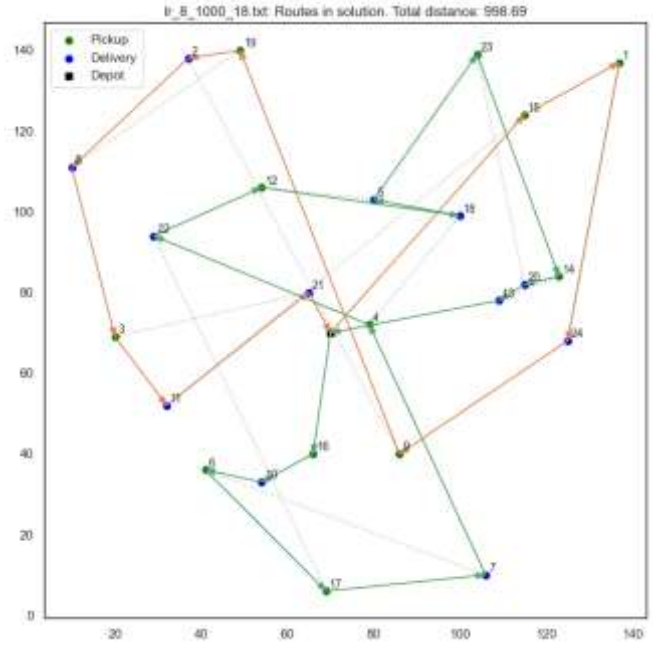

Рис. 4. Решение задачи с рис. 1–3. Последовательность стрелок одинакового цвета описывает один из маршрутов решения

Классической для задачи о приемке и доставке с временными ограничениями является 3-индексная формулировка [1] [19]. Вводятся 4 группы переменных:

- $x_{i,j}^k = 1$, если машина k следует из пункта i в пункт j, иначе 0 *(три индекса: два пункта и машина);*
- $y_{k,r} = 1$, если запрос r выполняется k-й машиной, иначе 0;
- S_i показывает время прибытия в пункт $i \in V$ ($S_0 = 0$);
- Q_i показывает загрузку машины после выезда из пункта $i \in V$, ($Q_0 = Q_{2n+1} = 0$).

Задача о приемке и доставке с временными ограничениями формулируется следующим образом:

$$\text{minimize} \left\{ \sum_{k\in K} \sum_{i\in V} \sum_{j\in V} c_{i,j} x_{i,j}^k \right\} \quad (1)$$

с учетом ограничений:

$$\sum_{j\in V\setminus\{i\}} x_{i,j}^k = y_{k,r(i)}, \quad k \in K, \ i \in N \quad (2)$$

$$\sum_{j\in V\setminus\{i\}} x_{j,i}^k = y_{k,r(i)}, \quad k \in K, \ i \in N \quad (3)$$

$$\sum_{j\in N} x_{0,j}^k \le 1, \quad k \in K \quad (4)$$

$$\sum_{k\in K} y_{k,r} = 1, \quad r \in R \quad (5)$$

$$S_j \ge S_i + s_i + t_{i,j} - M\left(1 - \sum_{k\in K} x_{i,j}^k\right), \quad i,j \in V \quad (6)$$

$$a_i \le S_i \le b_i, \quad i \in V \quad (7)$$

$$S_{d_r} \ge S_{p_r} + s_{p_r} + t_{p_r,d_r}, \quad r \in R \quad (8)$$

$$Q_j \ge Q_i + q_j - Q_{max}\left(1 - \sum_{k\in K} x_{i,j}^k\right), \quad i,j \in V \quad (9)$$

$$x_{i,j}^k, y_{k,r} \in \{0,1\}, \quad i,j \in V, \ k \in K, \ r \in R \quad (10)$$

$$0 \le S_i \le M, \quad i \in V \quad (11)$$

$$0 \le Q_i \le Q_{max}, \quad i \in V \quad (12)$$

Подробное описание ограничений даётся в 1-й части статьи [1]. Целевая функция (1) минимизирует общие транспортные расходы. M – достаточно большое число, которое может быть задано максимальным временем возврата в депо b_{2n+1}.

Опционально [19], в целевую функцию можно добавить стоимость F_k использования в решении дополнительной машины, тогда целевая функция будет иметь вид (где сумма по j для $x_{0,j}$ показывает задействована очередная машина или нет):

$$\text{minimize} \left\{ \sum_{k\in K} \sum_{i\in V} \sum_{j\in V} c_{i,j} x_{i,j}^k + \sum_{k\in K} F_k \sum_{j\in N} x_{0,j}^k \right\} \quad (1^*)$$

3-индексная формулировка имеет существенное достоинство: она явным образом описывает взаимодействие ключевых объектов в задаче в формулировке *«k-я машина следует из пункта i в пункт j»*. Но несмотря на то, что существуют и другие формулировки, даже с меньшим количеством оптимизируемых переменных, точные методы при больших размерностях задачи неэффективны, что вынуждает пользоваться эвристическими методами.

IV. Локальный поиск

Наиболее популярным эвристическим методом решения задач VRP является локальный поиск *(local search)*. Он основан на идее итеративного улучшения решения путём последовательного исследования его окрестности. **Окрестностью решения** называют множество выполнимых «соседних» решений, полученных путём применения элементарных операций к текущему решению.

В рамках задачи PDPTW, опишем основные этапы локального поиска. Он, как и многие эвристические методы, требует построения начального выполнимого решения. В данной статье элементарными операциями будем считать:

- Перемещение запроса из одного маршрута в другой *(PD-Shift, PD-Relocate)*: на рис. 5 показано перемещение запроса с пунктами $p_1 \to d_1$ из 1-го маршрута во 2-й;

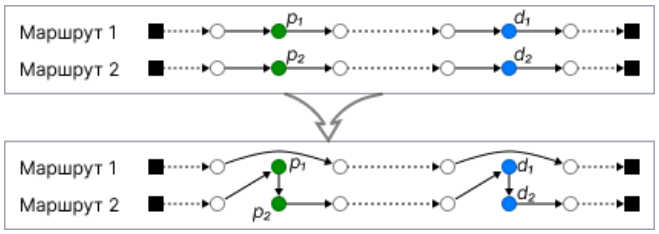

Рис. 5. Логика перемещения запроса из маршрута в маршрут

- Обмен двумя запросами между двумя маршрутами *(PD-Exchange):* на рис. 6 запрос с пунктами $p_1 \to d_1$ переносит из 1-го маршрута во 2-й, а запрос с пунктами $p_2 \to d_2$ из 2-го в 1-й;

- Перестановка запроса внутри одного маршрута *(PD-Rearrange, на рис. 7)*.

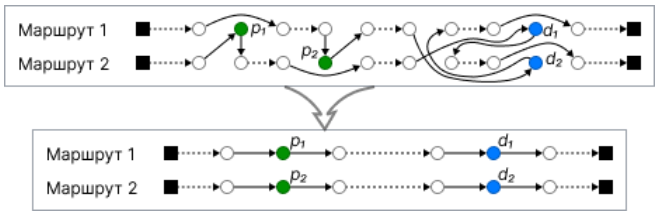

Рис. 6. Логика обмена запросами между маршрутами

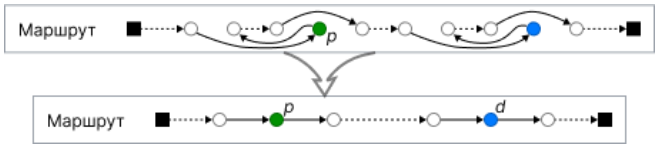

Рис. 7. Логика перестановки запроса внутри маршрута

Как было сказано ранее, для текущего решения S каждая такая элементарная операция порождает соответствующую окрестность: $O_{shift}(S)$, $O_{exchange}(S)$, $O_{rearrange}(S)$. На каждой итерации в окрестности текущего решения ищется решение с наименьшим значением целевой функции. Процесс продолжается до тех пор, пока не будет выполнен критерий останова (отсутствуют улучшения, превышено число итераций или время поиска и т.п.)

Как правило, все элементарные операции – «жадные», то есть направлены на максимальное уменьшение целевой функции, либо на её минимальное увеличение (то есть если удаляется запрос, то наиболее затратный, а если вставляется обратно – то так, чтобы общая стоимость решения[2] увеличилась минимально).

Ниже представлен алгоритм локального поиска.

Основные этапы локального поиска
1. Инициализация
• Сгенерировать начальное выполнимое решение S • $S_{best} = S$
2. Цикл итераций *(повторяется до выполнения критерия останова)*
• Провести поиск по окрестности $O_{shift}(S) \cup O_{exchange}(S)$ (т.е. рассмотреть все варианты перемещений и обменов запросов между маршрутами и выбрать наилучший) ⇒ получаем решение S' • *(Опционально)* Найти маршрут с минимальным количеством запросов, удалить его, а запросы переназначить в другие маршруты так, чтобы общая стоимость решения увеличилась минимально ⇒ получаем решение S'' • Провести поиск по окрестности $O_{rearrange}(S'')$ (т.е. ищется запрос с наибольшими затратами на его выполнение, и в маршруте, где он находится производятся перестановки пунктов с целью уменьшения затрат) ⇒ получаем решение S''' • $S_{new} = S'''$ *(новое решение)* **Оценка**: вычисляется стоимость нового решения $f(S_{new})$
• Если $f(S_{new}) < f(S_{best})$, то $S_{best} = S_{new}$ • $S = S_{new}$

Несмотря на простоту алгоритма, классический локальный поиск обладает известным недостатком: он склонен к «застреванию» в локальных экстремумах, которые могут быть далеки от глобального. Для преодоления этой проблемы разработано множество расширений локального поиска, таких как метод имитации отжига *(Simulated Annealing) [7]*, поиск с запретами *(Tabu Search) [11]*, направленный поиск *(Guided Local Search) [12]*, в котором повторные решения или решения с общими маршрутами штрафуются, и другие методы. Таким образом, сама логика итеративного поиска нового решения служит основной для более продвинутых алгоритмов и эвристик.

V. АДАПТИВНЫЙ ПОИСК В БОЛЬШИХ ОКРЕСТНОСТЯХ

Адаптивный поиск в больших окрестностях (ALNS) можно рассматривать как обобщение и развитие идей локального поиска. Он достаточно хорошо описан в статьях [5] [6], и показал себя как мощную и гибкую метаэвристику. Однако авторы рассматривают очень малый набор операторов, в этой статье этот набор значительно расширен.

В классическом локальном поиске, как было сказано, улучшение решения достигается за счёт применения элементарных операций, которые изменяют решение в пределах малой окрестности, то есть, в случае задачи PDPTW, производятся операции только с одним или двумя запросами в решении. В ALNS предлагается все элементарные операции упростить до двух видов:

- Разрушение решения *(Destroy)*, т.е. удаление запросов из некоторых маршрутов
- Восстановление решения *(Repair)*, т.е. вставка удаленных запросов в новые позиции в маршрутах

И тогда, в рамках ALNS, окрестностью текущего решения будет считаться множество решений, которые могут быть получены путём применения пары операторов разрушения и восстановления к исходному решению, и эта окрестность задаётся множеством подокрестностей решений, порожденных всеми возможными комбинациями операторов разрушения и восстановления *(декартово произведение)*.

Кроме того, ALNS включает в себя механизм адаптивного выбора операторов, что добавляет в алгоритм некий элемент обучающей стратегии — результаты применения операторов на предыдущих операторов влияют на вероятность их выбора в будущем, то есть алгоритм не просто последовательно или случайным образом выбирает определенные операторы разрушения или восстановления, он во время поиска подстраивается под определённый пример задачи, выбирая ту пару операторов разрушения и восстановления, которые быстрее всего стремятся к экстремуму.

В совокупности все это позволяет эффективно «перескакивать» через локальные экстремумы и значительно расширить область поиска по сравнению с классическим локальным поиском. Благодаря этим достоинствам ALNS получил широкое применение в таких областях, как маршрутизация транспорта, планирование и логистика.

[2] Под стоимостью решения имеется в виду целевая функция

Ниже представлен алгоритм адаптивного поиска в больших окрестностях, используемого в задаче о приёмке и доставке:

Основные этапы ALNS

1. Инициализация

- Сгенерировать начальное выполнимое решение S
- Задать список операторов разрушения $DO = \{do_1 \dots, do_m\}$ и восстановления $IO = \{io_1, \dots, io_k\}$
- Сгенерировать одинаковые вероятности выбора операторов
- $S_{best} = S$

2. Цикл итераций *(повторяется до выполнения критерия останова)*

- Копируется текущее решение $S_{temp} = S$
- Случайным образом выбирается число q – количество запросов, которые будут удаляться и вставляться
- **Выбор операторов**: случайным образом выбираются операторы разрушения $do \in DO$ и восстановления $io \in IO$ с вероятностями, пропорциональными их текущим весам
- **Разрушение**: оператор do применяется к текущему решению S. Удаляются запросы $r_1, \dots, r_q$, получается решение S'
- **Восстановление**: оператор io применяется к «разрушенному» решению S'.
  Запросы $r_1, \dots, r_q$ вставляются обратно, получается решение S''
- $S_{new} = S''$ *(новое решение)*
- **Оценка**: вычисляется стоимость нового решения $f(S_{new})$
  - Если $f(S_{new}) < f(S_{best})$, то $S_{best} = S_{new}$
  - Если выполнен *критерий принятия решения* S_{new} с учетом S_{temp}, то $S = S_{new}$, иначе $S = S_{temp}$
- **Адаптация весов**: Веса операторов do и io обновляются на основе их «успешности» (например, при улучшении текущего решения или нахождении глобально лучшего решения, вероятность выбора в будущем возрастает)

5.1. Построение начального решения

ALNS требует построения начального выполнимого решения. В нашей реализации алгоритма его предлагается строить с помощью случайных выполнимых вставок, либо с помощью «жадных» вставок, при которых стоимость решения увеличивается минимальным образом. Порядок вставки запросов определяется случайным образом. В пункте 5.4 будет описана логика вставки запроса в маршрут.

На рис. 8 показан пример задачи о приемке и доставке, на рис. 9–11 показана последовательность вставок запросов для формирования начального выполнимого решения:

- 1-й запрос вставился как есть, образуя маршрут: «депо → 1 → 6 → депо»;
- 2-й запрос вставился таким образом, что он исполняется одновременно с 1-м запросом: «депо → 1 → 2 → 6 → 7 → депо»;
- 3-й запрос идёт после пункта доставки 2-го запроса: «депо → 1 → 2 → 6 → 7 → 3 → 8 → депо» и т.д.

Похожим образом вставляются оставшиеся запросы. На рис. 12 (снизу) показано сформированное по такому принципу начальное выполнимое решение.

На практике проводились эксперименты с выбором способа построения начального выполнимого решения. Забегая вперед скажем, что в большинстве случаев «жадные» вставки строили решения с меньшей целевой функцией, чем у решений, построенных случайными вставками (что и неудивительно, однако случайные вставки помогают строить такие начальные точки, из которых можно не попадать в локальные минимумы).

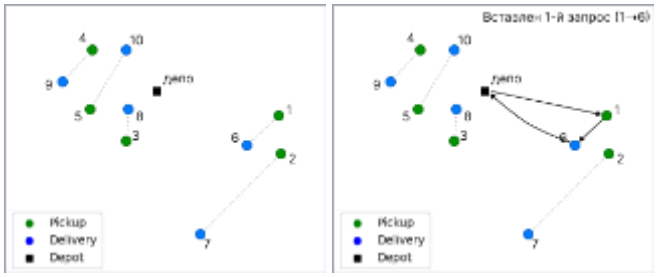

Рис. 8–9. Слева – пример задачи о приемке и доставке (обозначения как на рис. 4). Справа – вставка 1-го запроса при создании начального выполнимого решения

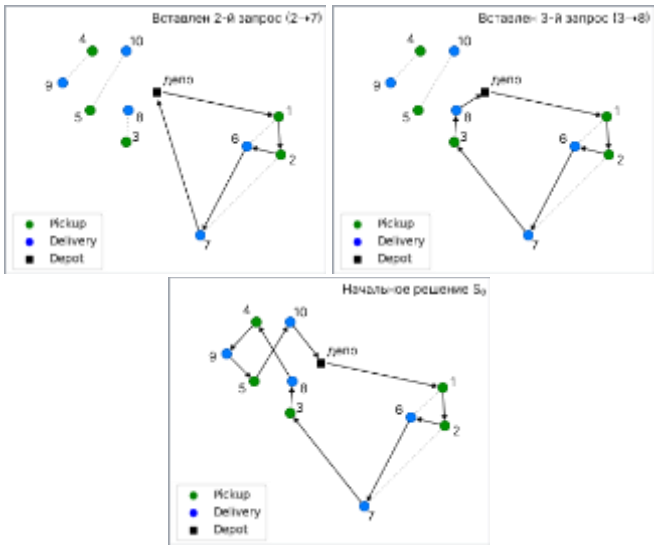

Рис. 10–12. Последовательные вставки запросов при создании начального выполнимого решения. Снизу – полный вариант начального решения для рис. 8 (слева)

5.2. Выбор операторов

На каждой итерации вероятностным образом выбираются оператор разрушения do и оператор восстановления io:

- С помощью оператора разрушения do в текущем решении определяются запросы и позже удаляются из него (рис. 13);

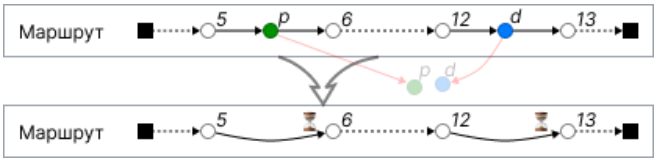

Рис. 13. Удаление запроса из маршрута

- Затем с помощью оператора io эти запросы вставляются обратно (на те же или на новые позиции), образуя новое решение (рис 14).

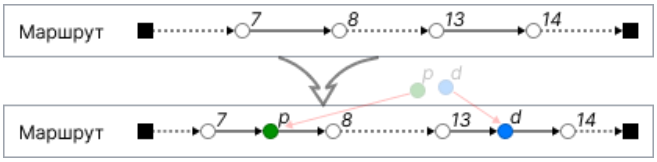

Рис. 14. Выполнимая вставка запроса в маршрут

В ALNS большую роль играет размер окрестности. Явным образом её размер задать нельзя, но можно задать границы для числа q – числа запросов, которые будут удаляться из решения и вставляться обратно. Нижняя граница числа q равняется единице, а верхняя граница контролируется гиперпараметром *«доля удаляемых запросов»* (5%, 10%, 20%), который домножается на число запросов в задаче. В нашей реализации алгоритма, число q на каждой итерации случайным образом выбирается между нижней и верхней границей. Стоит отметить, что с увеличением числа q растёт и время поиска решения.

5.3. Операторы разрушения

«Разрушение» решения подразумевает удаление запросов из маршрутов решения: то есть, например для запроса r, связывающего пункты $p_r \to d_r$ в некотором маршруте, подразумевается, что из него исключается посещение этих пунктов. Маршрут при этом остается выполнимым, так как нарушений каких-либо ограничений в таком случае не возникает.

Однако в «разрушенном» решении может возникнуть ситуация с длительным ожиданием перед пунктами, следующими за пунктами приёмки или доставки удаленного запроса. Для преодоления такой проблемы в статьях [6] [13] предлагается добавить в целевую функцию штраф на длительные ожидания перед пунктами. Но в нашей реализации предлагается создать два отдельных оператора разрушения, которые будут удалять запросы с наибольшим суммарным ожиданием перед пунктами, и наибольшим суммарным опозданием относительно начала временных окон клиентских пунктов. Поскольку поиск – итеративный процесс, то с некоторой вероятностью эти операторы будут выбраны на следующих итерациях и маршруты будут перестроены.

Кроме того, в нашей реализации используются некоторые операторы разрушения, описание которых уже встречается в научной литературе. Их условно можно разделить на две группы:

- «Worst-of» операторы: при удалении запроса стоимость (общее время, или суммарное время ожидания) решения уменьшается максимально;
- Операторы, разрушающие локальные связи, они же эвристики Шэу [15] *(Shaw heuristics)*: такие операторы удаляют те запросы, которые максимально близко находятся друг к другу, имеют похожие временные интервалы обслуживания или вес груза. Такие эвристики направлены не на поиск каких-то отдельных «плохих» запросов, а на разрушение целых локальных кластеров запросов, тем самым перескакивая через локальные минимумы.

В данной статье используются следующие «worst-of» операторы разрушения (в приложении [15] к статье имеется их описание):

- Удаление случайных запросов *(random request removal)*;
- Удаление запросов, которые наибольшим образом сокращают длину маршрута *(worst cost request removal);*
- Удаление запросов с наибольшим опозданием после начала временного окна у пунктов приемки и доставки *(worst time delay request removal)*;
- Удаление запросов с наибольшим ожиданием перед пунктами приемки и доставки *(worst time wait request removal)*;
- Выбор случайного маршрута и удаление из него запроса, вносящего наибольший вклад в длину маршрута *(worst request in random route removal)*;
- Удаление запросов, чей вклад в длину маршрута максимальный (на примере рис. 13, под вкладом подразумевается соотношение $\rho_{5,p} + \rho_{p,6} + \rho_{12,d} + \rho_{d,13}$ к общей длине маршрута);
- Удаление запросов с пунктами, имеющими самые узкие временные окна *(time window request removal).*

Описание операторов Шэу можно найти в статьях [13] [15]. В случае, когда в целевую функцию включена стоимость использования дополнительной машины в решении, то имеет смысл ввести оператор разрушения, заключающийся в полном удалении случайного маршрута или маршрута с минимальной длиной.

5.4. Операторы восстановления

После «разрушения» решения, необходимо провести его «восстановлениие». Как было сказано ранее, при «разрушении» выполнимого решения его маршруты продолжают оставаться выполнимыми. А вот при применении операторов восстановления это не всегда соблюдается. Во избежание построения невыполнимого решения, оператор восстановления необходимо дополнить процедурой проверки выполнимости *(feasibility)* нового решения: то есть сначала необходимо проверить все возможные варианты вставки на их выполнимость, а потом уже выбрать среди выполнимых вставок нужную. Если окажется, что запрос вставить в существующие маршруты нельзя без нарушений ограничений, тогда создается новый маршрут в решении (задействуется еще одна машина).

В научной литературе операторы восстановления описаны куда менее разнообразно, чем операторы разрушения, и в реализации использовано три вида операторов:

- «жадный» *(greedy insert)*: среди всех запросов ищет такую комбинацию запроса и позиции вставки пунктов приёмки и доставки этого запроса, которая минимально увеличивает общую стоимость решения;
- «сожалеющий» *(regret insert)*: среди всех запросов ищет тот, который имеет наибольшую разницу по стоимости вставки между двумя наилучшими позициями, и вставляет этот запрос на его наилучшую позицию;
- Случайный *(random insert)*: запрос вставляется в случайную выполнимую позицию.

5.5. Механизм адаптивных запретов

Это небольшое новшество для алгоритма ALNS сделано для повышения его эффективности. Основная цель заключается в предотвращении повторяющихся итераций, ведущих к бесполезному расходованию вычислительных ресурсов. Проблема возникает из-за того, что один и тот же запрос регулярно удаляется и вставляется на одну и ту же позицию в ходе последовательных итераций. Для её устранения предложены два отдельных набора запретов:

- Запрет на удаление: запоминает конкретные варианты удаления запросов из маршрутов
- Запрет на вставку: фиксирует нежелательные варианты вставки запросов обратно в маршруты

Каждый вариант удаления или вставки подразумевает под собой комбинацию трех элементов:

- запроса,
- позиций вставки пунктов приёмки и доставки
- и части маршрута, куда будет вставляться или откуда будет удаляться запрос, от начала до пункта доставки

По мере прохождения итераций уже использованные варианты удаления или вставки будут записываться в соответствующие наборы. Однако, чтобы избежать чрезмерного ограничения по вариантов вставки или удаления, используется принцип очереди *(«FIFO», first-in-first-out)*, что означает, что самые старые варианты будут исключаться из набора запретов спустя некоторое количество итераций.

Таким образом, при потенциальной «жадной» вставке или «worst-of» удалении запроса запрещенные варианты рассматриваться не будет, тем самым вынуждая алгоритм искать другие решения.

5.6. Оценка решения, адаптация весов и гиперпараметры алгоритма

После манипуляций с решением необходимо провести его оценку. Если значение целевой функции для нового решения оказывается меньше, чем для наилучшего решения, то новое решение устанавливается в качестве наилучшего. Дополнительно проверяется *критерий принятия решения*, заключающийся в том, что новое решение S_{new} с учетом текущего решения S принимается с вероятностью $e^{-(f(S_{new})-f(S))/T}$, где T – «температура», которая уменьшается на каждой итерации, как это происходит в алгоритме имитации отжига.

С помощью *гиперпараметра температуры* τ и стоимости начального решения вычисляется *стартовая температура* T_0 (также позаимствовано из алгоритма имитации отжига):

$$T = -\frac{\tau * f(S_0)}{\ln(0.5)}$$

Далее с каждой новой итерацией температура будет уменьшаться на значение *гиперпараметра охлаждения* λ *(например, 0.1%)*: $T'' = T(1 - \lambda)$

Критерием останова может быть:

- ограничение по времени выполнения алгоритма (например, 1 час);
- ограничение по числу итераций;
- или отсутствие улучшений лучшего решения на протяжении некоторого числа итераций (например, на протяжении 50 итераций при общем числе в 1000 итераций целевая функция наилучшего решения уменьшилась меньше чем на 0.5%).

В конце итерации происходит обновление весов операторов, которые влияют на вероятность выбора того или иного вида оператора разрушения или восстановления. В работе вес оператора зависит от:

- *частоты выбора* этого оператора *(count)* на предыдущих итерациях,
- *показателя результативности (score)* этого оператора в течение выполнения алгоритма
- *гиперпараметра реакции, обучения (reaction factor, decay)*, контролирующего изменение веса оператора в зависимости от результативности на последних итерациях *(в работе использовались значения от 5 до 35%)*. Он определяет, насколько оперативно алгоритм реагирует на текущую эффективность операторов.

На 1-й итерации *score* у всех операторов равен нулю, и они увеличиваются с каждой итерацией в соответствии со следующим правилом:

- Если новое решение – глобально наилучшее, то *score* этих операторов увеличивается на 1.5
- Если новое решение лучше текущего, то *score* увеличивается на 1.2
- Если новое решение с учетом текущего принимается *(имитация отжига)*, то *score* растёт на 0.1
- Если не принимается, то *score* растёт на 0.5

После обновления показателей результативности обновляются веса эвристик по формуле:

$$weight'' = weight\ (1 - decay) + \frac{score}{count}$$

При вероятностном выборе оператора веса нормируются так, чтобы их сумма равнялась единице.

VI. Численное исследование

В нашей статье для проведения анализа используется датасет Li Lim [20]. В нем имеются примеры задач для 100, 200 и 400 клиентских пунктов. Дополнительно были сгенерированы задачи с 25 и 50 пунктами, случайно отобранными из задач с большими размерностями по алгоритму, описанному в 1-й части статьи [1]. Все примеры задач в датасете содержат информацию о местоположении пунктов, их типе (приёмка / доставка / депо) и их временных интервалов обслуживания. Датасет поделен на три части в зависимости от того, как расположены клиентские пункты на карте: LC (, LR и LRC (рис. 15–17). Клиенты в задачах LC распределены группами по карте. В задачах LR клиенты распределены случайным образом равномерно по всей карте. В задачах LRC клиенты наполовину сгруппированы и наполовину распределены по карте. В задачах из датасета Li Lim считается, что расстояние и время поездки между двумя пунктами не зависит от направления, то есть $\rho_{i,j} = \rho_{j,i}$ и $t_{i,j} = t_{j,i}$ для любого $i, j \in V$.

Дополнительно был использован датасет Sartori [21], он похож на Li Lim, но в нем примеры задачи сгенерированы на основе карты городов, этот датасет берет в основу реальные кейсы доставок и учитывает, что расстояние и время в пути между пунктами может зависеть от направления. Похожие примеры можно сгенерировать применив шум к расстояниям.

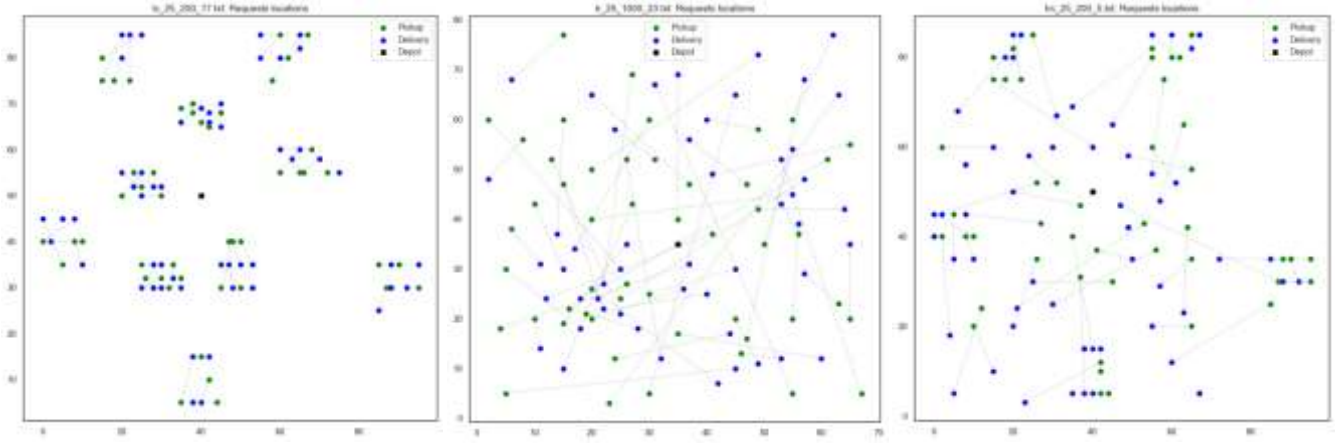

Рис. 15–17. Примеры задач из датасета Li Lim. Слева направо: LC (клиенты расположены группами по карте), LR (клиенты равномерно распределены по карте) и LRC (сочетание группировки и случайного распределения)

Код был реализован на Python [23] с использованием библиотек Numpy [24], Scipy [25] и Joblib [22]. Эксперименты проводились на Apple M1 Pro (ARM, 8 ядер 3.22 ГГц, 16 Гб ОЗУ), ограничение по времени поиска – 1 час. При тестировании предлагалось выполнить 100, 500 и 1 000 итераций. Максимальное число удаляемых запросов q устанавливалось в размере 5%, 15% и 25% от общего числа запросов.

Перейдем к анализу результатов.

Таблица 1. Статистика по средним распределения ошибок решений, полученных ALNS, относительно известных лучших решений [20] (для 25, 50 пунктов, относительно точного решения; прочерк – недостаточно данных для оценки)

Средние значения распределения ошибок решений										
Число пунктов и тип задачи		Кол-во итераций алгоритма ALNS								
		nbh_ratio = 0.05			*nbh_ratio = 0.15*			*nbh_ratio = 0.25*		
		100	500	1 000	100	500	1 000	100	500	1 000
24	LC	0.21%	0.20%	0.17%	0.74%	0.23%	0.08%	0%	0%	0%
	LR	0.65%	0.16%	0.35%	0.97%	0.59%	0.14%	0.09%	0%	0%
	LRC	0.31%	0.18%	0.03%	0.27%	0.14%	0.25%	0.04%	0%	0%
50	LC	7.21%	2.72%	1.53%	1.27%	0.58%	0.54%	0.86%	0.44%	0.44%
	LR	3.49%	0.60%	0.46%	0.76%	0%	0%	0%	0%	0%
	LRC	4.71%	1.19%	1.38%	0.76%	0%	0.05%	0.06%	0%	0%
100	LC	12.22%	1.11%	1.16%	1.22%	0.01%	0%	0.62%	0%	0%
	LR	19.00%	8.23%	4.73%	6.53%	1.25%	1.34%	4.03%	1.09%	0.40%
	LRC	20.00%	8.98%	6.63%	6.75%	1.54%	1.27%	4.15%	1.01%	0.87%
200	LC	23.25%	-1.38%	0.70%	5.17%	-0.60%	-1.08%	2.67%	-0.90%	-1.38%
	LR	15.09%	-6.29%	-1.19%	1.58%	-5.34%	-5.92%	-1.24%	-5.87%	-6.29%
	LRC	15.74%	-2.75%	1.41%	4.25%	-2.21%	-3.04%	1.71%	-2.18%	-2.75%

Ошибки варьируются в зависимости от типа задачи и её размерности: при 25, 50 пунктах они достигают практически нуля по отношению к точному решению. демонстрируя стабильность алгоритма. При больших размерностях (от 100 пунктов) ошибки возрастают. достигая 25%, что типично при росте размерности пространства решений и ограниченного времени на оптимизацию (табл. 1–2). LC-задачи показывают наиболее узкое распределение ошибок, для них в целом меньшее значение средней по ошибкам. меньший их разброс, но есть примеры с большими ошибками (т.н. выбросы). LR-задачи показывают более широкий разброс ошибок, с заметными выбросами и более высоким значением средней. LRC-задачи на больших размерностях имеют самое нестабильное распределение ошибок с множеством выбросов.

Таблица 2. Статистика по стандартным отклонениям распределений ошибок решений. полученных ALNS. относительно известных лучших решений [20] (для 25. 50 пунктов. относительно точного решения; прочерк – недостаточно данных для оценки)

Стандартные отклонения распределений ошибок решений										
Число пунктов и тип задачи		Кол-во итераций алгоритма ALNS								
		nbh_ratio = 0.05			*nbh_ratio = 0.15*			*nbh_ratio = 0.25*		
		100	500	1 000	100	500	1 000	100	500	1 000
24	LC	0.53%	0.45%	0.46%	1.76%	0.56%	0.22%	0%	0%	0%
	LR	1.30%	0.97%	1.09%	1.37%	1.35%	0.95%	0.34%	0%	0%
	LRC	0.55%	0.43%	0.11%	0.55%	0.38%	0.53%	0.40%	0.00%	0.06%
50	LC	7.32%	5.87%	5.92%	5.52%	5.08%	4.90%	5.19%	4.98%	4.93%
	LR	3.38%	2.92%	2.75%	3.21%	0%	0%	0%	0%	0%
	LRC	3.98%	2.40%	1.96%	2.15%	0%	2.24%	2.27%	0%	0%
100	LC	12.94%	4.01%	4.11%	4.23%	3.50%	3.50%	3.50%	3.50%	3.50%
	LR	6.91%	6.18%	4.47%	3.92%	2.06%	1.86%	3.79%	1.98%	1.38%
	LRC	8.99%	5.28%	3.91%	4.48%	2.25%	2.31%	3.61%	2.17%	2.18%
200	LC	7.56%	6.46%	4.56%	6.57%	3.81%	3.95%	5.76%	4.13%	3.83%
	LR	7.64%	6.99%	7.08%	7.40%	5.88%	5.54%	5.85%	5.89%	5.72%
	LRC	7.01%	6.04%	4.82%	6.10%	6.31%	6.29%	6.17%	6.51%	6.34%

Таблица 3. Статистика по времени поиска решения с помощью ALNS (прочерк – недостаточно данных для оценки)

Среднее время поиска решения (в сек.)										
Число пунктов и тип задачи		Кол-во итераций алгоритма ALNS								
		nbh_ratio = 0.05			*nbh_ratio = 0.15*			*nbh_ratio = 0.25*		
		100	500	1 000	100	500	1 000	100	500	1 000
24	LC	0.08	0.22	0.34	0.08	0.22	0.33	0.13	0.41	0.76
	LR	0.10	0.31	0.50	0.10	0.32	0.45	0.17	0.51	1.00
	LRC	0.08	0.27	0.41	0.09	0.24	0.37	0.15	0.45	0.78
50	LC	0.22	0.78	1.39	0.47	1.63	2.72	0.70	2.50	4.01
	LR	0.31	1.32	2.17	0.75	2.74	4.84	1.17	3.80	6.64
	LRC	0.28	1.05	1.94	0.64	2.27	3.93	1.00	3.49	5.75
100	LC	1.42	5.70	10.81	3.74	13.63	22.80	5.61	17.34	28.65
	LR	2.41	6.24	12.38	4.15	18.75	34.49	6.35	29.11	51.19
	LRC	2.55	5.32	9.90	3.45	16.01	28.66	5.60	23.98	44.53
200	LC	6.77	31.49	62.89	17.59	88.47	154.6	25.02	122.4	200.2
	LR	11.55	53.74	100.9	30.75	154.52	283.1	46.66	202.5	357.3
	LRC	9.16	46.25	85.48	27.03	123.07	230.9	37.12	187.2	327.0

Похожие рассуждения касательно и времени поиска: оно существенно возрастает с увеличением итераций, но по сравнению с методом ветвей и отсечений из 1-й части статьи [1], ALNS работает заметно быстрее, что указывает на хорошую масштабируемость алгоритма при ограниченных вычислительных ресурсах. LC-задачи решаются быстрее всего, а задачам типа LRC и LR требуется заметно больше времени (табл. 3).

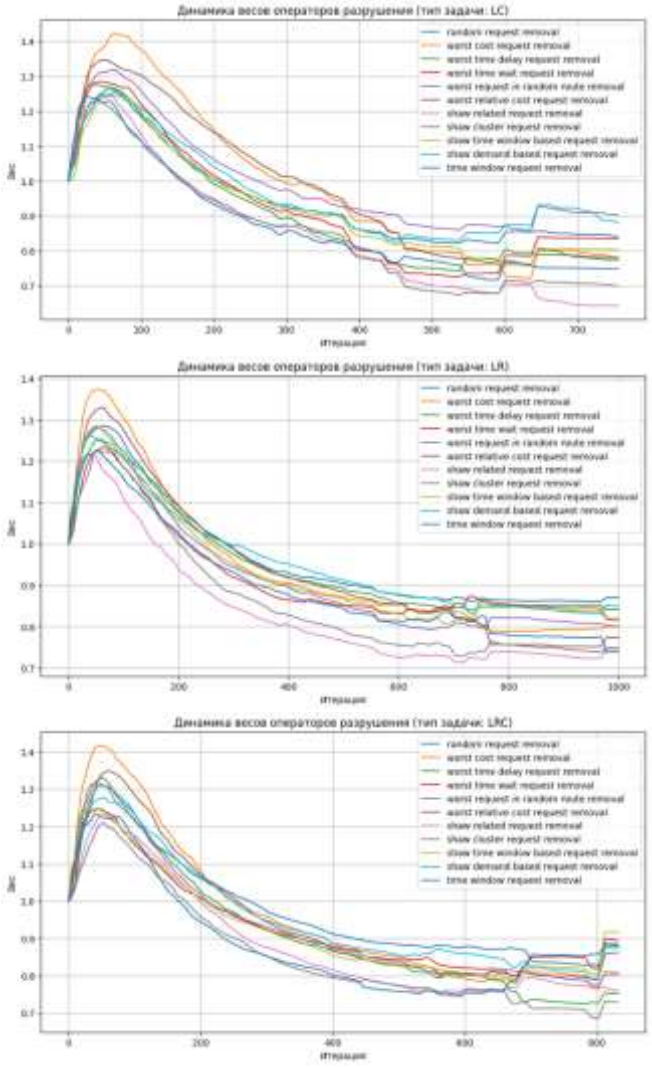

Рис. 18–20. Изменение весов операторов разрушения во время выполнения итераций в зависимости от типа задачи

Гиперпараметр (*nbh_ratio*) определяет долю решения, подлежащую разрушению на каждом шаге, что напрямую влияет на глубину поискового шага, а следовательно, на время и результаты поиска. Результаты анализа демонстрируют следующие закономерности (табл. 1–3):

- Малое значение *nbh_ratio* (0.05) в большинстве случаев обеспечивает наибольшую ошибку: решения слабо изменяются с ходом итераций
- Среднее значение (0.15) показывает смешанные результаты, с умеренным улучшением качества решения, но при этом время поиска увеличивается. В целом такое значение параметра обеспечивает наилучший баланс между скоростью и ошибкой.
- Высокое значение (0.25) приводит к наиболее агрессивному разрушению и последующему восстановлению решения. При этом ошибки становятся меньше, но время поиска растет значительно. Дальнейшее повышение значения *nbh_ratio* уже не имеет смысла, так как время одной итерации сильно возрастает, но при этом ошибки практически не уменьшаются.

Таким образом, *nbh_ratio* оказывает существенное влияние на эффективность ALNS. При прочих равных

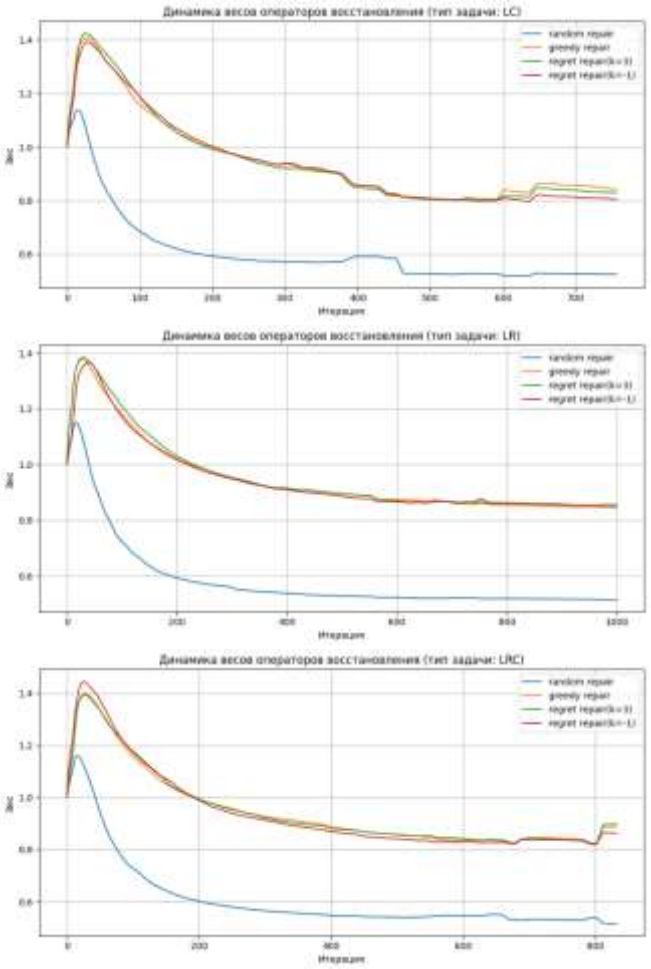

Рис. 21–23. Изменение весов операторов разрушения во время выполнения итераций в зависимости от типа задачи

условиях, значения в диапазоне 0.15–0.20 обеспечивают наилучший компромисс между глубиной поиска и его скоростью.

Теперь перейдем к анализу адаптивного изменения весов. На рис. 18–20 показано, как изменяются веса операторов разрушения в зависимости от типа задачи. И

- Для задач типа LC веса большинства операторов быстро расходятся. Наиболее эффективными являются операторы, удаляющие самые затратные запросы во всем решении или в случайном маршруте решения *(worst cost request removal, worst relative cost request removal, worst request in random route removal).*
- В LR-задачах веса распределяются более равномерно дольше, что может указывать на менее выраженные различия в эффективности операторов.
- В задачах типа LRC наблюдаются т.н. «скачки» весов, что указывает на высокую чувствительность к шуму и стохастическим аспектам инстансов. Наиболее

эффективным, неожиданно, оказался оператор, который удаляет запросы с самыми узкими временными окнами *(time window request removal)*.

Эвристики, связанные с опозданиями или ожиданиями *(time delay, time wait)*, показали средние результаты, что говорит о их периодическом применении из-за появляющихся существенных суммарных задержек или опозданий в решениях спустя несколько итераций алгоритма.

Эвристики Шэу часто демонстрировали низкую эффективность, у них самые низкие веса.

На рис. 21 – 23 показана динамика изменения весов операторов восстановления для разных типов задач. В целом, для каждого типа задачи ярко выражено доминирование жадных стратегий *(greedy_repair, regret repair)*. Вес оператора случайной вставки *(random repair)* не обнуляется полностью, вероятно потому, что наличие элемента случайности полезно для улучшения качества решений в условиях высокой размерности и неопределенности пространства возможных вариантов.

Что касается гиперпараметра обучения *(decay)*, то наилучший баланс между качеством и производительностью достигается при *decay* ≈ *0.1–0.2*. Низкие *decay (<0.1)* приводят к быстрой адаптации, но с риском застревания в локальных минимумах. Высокие значения *(> 0.5)* ухудшают конечное решение: история «успехов» операторов почти не учитывается, вес перескакивает от одного значения к другому.

VII. Заключение

В данной статье был рассмотрен эвристически подход к решению задачи о приемке и доставке с временными ограничениями. Был подробно описан алгоритм адаптивного поиска в больших окрестностях, внесены в него новшества, связанные с операторами опозданий и задержек по временным окнам, внедрен механизм адаптивных запретов, вынуждающий алгоритм искать новые решения.

В рамках численного исследования была выполнена комплексная оценка эффективности алгоритма на множестве задач различных типов (LC, LR, LRC). Результаты позволяют сделать несколько выводов.

Задачи типа LC (жёсткие ограничения) решаются быстрее, и с самой меньшей ошибкой. Задачи типа LR и особенно LRC демонстрируют меньшее качество решений и требуют значительно больше времени из-за стохастической природы данных.

Наиболее эффективными операторами разрушения оказались операторы, связанные со максимальным уменьшением стоимости решения (worst cost, worst relative cost). Параметр *decay* лучше всего помогает алгоритму адаптироваться к условиям задачи при значениях в диапазоне 0.1–0.2.

В целом ALNS показал себя как мощную и гибкую метавэристику. Алгоритм потенциально можно использовать при различных расширениях задач PDPTW с компонентой неизвестности, таких как Stochastic PDPTW с неопределёнными временными окнами или временем в пути; Dynamic PDPTW, где новые запросы поступают во время выполнения маршрута; или при задаче, где при построении маршрутов нужно учесть будущие запросы (т.е. учесть прогноз на основе истории запросов).

Библиография

[1] Царьков, Никита Владимирович, and Дмитрий Юрьевич Голембиовский. "Обзор методов решения задачи о приёмке и доставке с временными ограничениями. Часть I: точный подход." International Journal of Open Information Technologies 13.7 (2025): 60-70.

[2] Toth, P., & Vigo, D. (2002). An Overview of Vehicle Routing Problems. The Vehicle Routing Problem, 1–26.

[3] Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research, 35(2), 254–265.

[4] Lau, H. C. W., & Liang, Z. (2002). Pickup and delivery problem with time windows using a hybrid genetic algorithm. Proceedings of the IEEE International Conference on Robotics and Automation, 17(3), 1470–1475.

[5] Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. Computers & Operations Research, 34(8), 2403–2435.

[6] Ropke, S., & Pisinger, D. (2006). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. Transportation Science, 40(4), 455–472.

[7] Li, H., & Lim, A. (2003). A metaheuristic for the pickup and delivery problem with time windows. Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence, 160–167.

[8] Nanry, W. P., & Barnes, J. W. (2000). Solving the pickup and delivery problem with time windows using reactive tabu search. Transportation Research Part B: Methodological, 34(2), 107–121.

[9] Montemanni, R., Gambardella, L. M., Rizzoli, A. E., & Donati, A. V. (2005). Ant colony system for a dynamic vehicle routing problem. Journal of Combinatorial Optimization, 10(4), 327–343.

[10] Harbaoui, M. H., & Kammarti, R. (2010). Multi-objective optimization of the dynamic pickup and delivery problem with time windows using a genetic algorithm.

[11] Semet, F., & Taillard, E. (1993). Solving real-life vehicle routing problems efficiently using tabu search. Annals of Operations Research, 41(4), 469–488.

[12] Laporte, G. (2009). Fifty Years of Vehicle Routing. Transportation Science, 43(4), 408–416.

[13] Mancini, S. (2016). A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based methaeuristic. Transportation Research Part C: Emerging Technologies, 70, 100–112.

[14] Cordeau, J.-F. & Maischberger, M. (2012). A parallel iterated tabu search heuristic for vehicle routing problems. Computers & Operations Research, 39(9), 2033–2050.

[15] P. Shaw. (1998). Using constraint programming and local search methods to solve vehicle routing problems. International Conference on Principles and Practice of Constraint Programming, 417–431.

[16] Li, H & Lim, A. (2001). A metaheuristic for the pickup and delivery problem with time windows. International Journal of Artificial Intelligence Tools, 12(2), 160–170.

[17] Fang, Y., Lin, X., & Luo, J. (2024). Learning-based heuristics for pickup and delivery with time windows. Under Review.

[18] Rabecq, C., & Chevrier, V. (2022). A deep learning architecture based on attention mechanisms for dynamic pickup and delivery problems.

[19] Ropke, S. & Cordeau, J.-F. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks, 49(4), 258-272.

[20] Набор данных Li & Lim Benchmark. [Электронный ресурс]. URL: https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/ (дата обращения: 01.02.2025)

[21] Sartori, C. S., & Buriol, L. S. (2020). Instances for the Pickup and Delivery Problem with Time Windows based on open data. Mendeley Data, V2. [Электронный ресурс]. URL: https://data.mendeley.com/datasets/wr2ct4r22f/2 (дата обращения 01.02.2025)

[22] Приложение. Результаты расчетов. [Электронный ресурс]. URL: https://docs.google.com/spreadsheets/d/1Dw42k1hHC4vJEfgBg6Q7IV4XkfCALOP_t6QpClgOuoA/edit?usp=sharing (дата обращения: 01.02.2025)
[23] Python. [Электронный ресурс]. URL: https://www.python.org/ (дата обращения: 01.02.2025)
[24] Numpy. [Электронный ресурс]. URL: https://numpy.org (дата обращения: 01.02.2025)
[25] Scipy. [Электронный ресурс]. URL: https://scipy.org (дата обращения: 01.02.2025)
[26] Joblib. [Электронный ресурс]. URL: https://joblib.readthedocs.io/en/stable/ (дата обращения: 01.02.2025)

On methods for solving pickup and delivery problem with time windows. Part II: heuristic approach

N. Tsarkov, D. Golembiovsky

***Abstract* — This paper focuses on the Pickup and Delivery Problem with Time Windows (PDPTW), an optimization problem arising in numerous logistics and transportation settings. To address this problem, we propose a variant of the Adaptive Large Neighborhood Search (ALNS) metaheuristic, which integrates a diverse set of problem-specific destroy and repair operators. The algorithm is further enhanced with a tabu-based memory mechanism to prevent cycling and improve exploration of the solution space. The proposed method demonstrates a favorable balance between solution quality and computational efficiency. Moreover, its modular structure makes it well-suited for adaptation to various extensions of the PDPTW, including stochastic and dynamic scenarios. Computational experiments on benchmark datasets from Li & Lim and Sartori confirm the effectiveness and robustness of the approach.**

***Keywords* — VRP, PDPTW, ILP, ALNS, Local Search, heuristics, metaheuristics, tabu search, vehicle routing.**

REFERENCES

[1] Tsarkov, Nikita, and Dmitry Golembiovsky. "On methods for solving pickup and delivery problem with time windows. Part I: exact approach." International Journal of Open Information Technologies 13.7 (2025): 60-70.

[2] Toth, P., & Vigo, D. (2002). An Overview of Vehicle Routing Problems. The Vehicle Routing Problem, 1–26.

[3] Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research, 35(2), 254–265.

[4] Lau, H. C. W., & Liang, Z. (2002). Pickup and delivery problem with time windows using a hybrid genetic algorithm. Proceedings of the IEEE International Conference on Robotics and Automation, 17(3), 1470–1475.

[5] Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. Computers & Operations Research, 34(8), 2403–2435.

[6] Ropke, S., & Pisinger, D. (2006). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. Transportation Science, 40(4), 455–472.

[7] Li, H., & Lim, A. (2003). A metaheuristic for the pickup and delivery problem with time windows. Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence, 160–167.

[8] Nanry, W. P., & Barnes, J. W. (2000). Solving the pickup and delivery problem with time windows using reactive tabu search. Transportation Research Part B: Methodological, 34(2), 107–121.

[9] Montemanni, R., Gambardella, L. M., Rizzoli, A. E., & Donati, A. V. (2005). Ant colony system for a dynamic vehicle routing problem. Journal of Combinatorial Optimization, 10(4), 327–343.

[10] Harbaoui, M. H., & Kammarti, R. (2010). Multi-objective optimization of the dynamic pickup and delivery problem with time windows using a genetic algorithm.

[11] Semet, F., & Taillard, E. (1993). Solving real-life vehicle routing problems efficiently using tabu search. Annals of Operations Research, 41(4), 469–488.

[12] Laporte, G. (2009). Fifty Years of Vehicle Routing. Transportation Science, 43(4), 408–416.

[13] Mancini, S. (2016). A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based methaeuristic. Transportation Research Part C: Emerging Technologies, 70, 100–112.

[14] Cordeau, J.-F. & Maischberger, M. (2012). A parallel iterated tabu search heuristic for vehicle routing problems. Computers & Operations Research, 39(9), 2033–2050.

[15] P. Shaw. (1998). Using constraint programming and local search methods to solve vehicle routing problems. International Conference on Principles and Practice of Constraint Programming, 417–431.

[16] Li, H & Lim, A. (2001). A metaheuristic for the pickup and delivery problem with time windows. International Journal of Artificial Intelligence Tools, 12(2), 160–170.

[17] Fang, Y., Lin, X., & Luo, J. (2024). Learning-based heuristics for pickup and delivery with time windows. Under Review.

[18] Rabecq, C., & Chevrier, V. (2022). A deep learning architecture based on attention mechanisms for dynamic pickup and delivery problems.

[19] Ropke, S. & Cordeau, J.-F. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks, 49(4), 258-272.

[20] Набор данных Li & Lim Benchmark. [Online]. URL: https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/ (Available 01.02.2025)

[21] Sartori, C. S., & Buriol, L. S. (2020). Instances for the Pickup and Delivery Problem with Time Windows based on open data. Mendeley Data, V2. [Online]. URL: https://data.mendeley.com/datasets/wr2ct4r22f/2 (Available 01.02.2025)

[22] Приложение. Результаты расчетов. [Электронный ресурс]. URL: https://docs.google.com/spreadsheets/d/1Dw42k1hHC4vJEfgBg6Q7IV4XkfCALOP_t6QpClgOuoA/edit?usp=sharing (дата обращения: 01.02.2025)

[23] Python. [Online]. URL: https://www.python.org/ (Available 01.02.2025)

[24] Numpy. [Online]. URL: https://numpy.org (Available 01.02.2025)

[25] Scipy. [Online]. URL: https://scipy.org (Available 01.02.2025)

[26] Joblib. [Online]. URL: https://joblib.readthedocs.io/en/stable/ (Available 01.02.2025)

Manuscript received June 13, 2025.

Nikita Tsarkov, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University (e-mail: tsarkov90@gmail.com).

Golembiovsky Dmitry, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University (email: golemb@cs.msu.ru)