ДИСКРЕТНЫЙ АНАЛИЗ И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ
Ноябрь—декабрь 2015. Том 22, № 6. С. 55–77

УДК 519.85 DOI: 10.17377/daio.2015.22.496

ТРЁХФАЗНЫЙ АЛГОРИТМ ОПТИМИЗАЦИИ АВТОПАРКА И МАРШРУТОВ ТРАНСПОРТНЫХ СРЕДСТВ *)

А. В. Хмелёв[1]
[1]Новосибирский гос. университет,
ул. Пирогова, 2, 630090 Новосибирск, Россия
e-mail: avhmel@gmail.com

Аннотация. Рассматривается задача оптимизации автопарка и маршрутов транспортных средств в предположении, что каждый клиент имеет временно́е окно для его обслуживания. Водители транспортных средств работают посменно. Каждая смена имеет начало, конец и определённое число перерывов для отдыха. Построена математическая модель в терминах частично-целочисленного линейного программирования. Разработан трёхфазный алгоритм локального поиска с эффективной процедурой просмотра окрестности. Численные эксперименты на тестах одной из новосибирских транспортных компаний показали эффективность разработанного подхода и значительное снижение издержек. Табл. 2, ил. 4, библиогр. 13.

Ключевые слова: задача маршрутизации, временно́е окно, рабочая смена, перерыв, локальный поиск, оптимизация автопарка.

Введение

Рассматривается задача минимизации затрат на доставку грузов заданному множеству клиентов. Каждый клиент имеет временно́е окно, в котором должно начаться его обслуживание. Известна длительность обслуживания и объём доставляемого груза. Транспортные средства (ТС) имеют различную грузоподъёмность. За каждым ТС закреплён водитель, который работает в определённую рабочую смену, например, с 7:00 до 16:00 или с 13:00 до 23:00. В ходе работы водитель обязан делать перерывы на отдых. Для перерывов заданы длительность и временны́е окна. Требуется построить маршруты для ТС и доставить грузы всем клиентам так, что суммарные затраты на привлечённый автопарк и перевозку минимальны.

*)Исследование выполнено при финансовой поддержке Российского фонда фундаментальных исследований (проект 15–07–01141).

Задача маршрутизации с временны́ми окнами клиентов активно исследуется последние 15–20 лет [4, 5]. Предложен широкий спектр различных моделей, среди которых есть и модели оптимизации автопарка [2, 7]. Одна из наиболее эффективных эвристик для этих задач представлена в [12]. Технологические перерывы в работе водителей (паузы) также рассматривались в моделях маршрутизации [3, 13]. Такие требования чаще всего возникают при перевозках на большие расстояния [8–10]. В настоящей работе предполагается, что все клиенты находятся в одном городе, их много и больших переездов нет. В [6] построена математическая модель частично-целочисленного линейного программирования с одной паузой в каждом маршруте. Эта пауза должна идти сразу после обслуживания клиента. Такое требование сильно сужает область применимости модели. В настоящей работе построена новая модель с произвольным числом пауз, а их начало может быть любым в рамках заданных временны́х окон. Получены точные значения для начала пауз при фиксированной последовательности обхода клиентов. Для оптимизации автопарка разработана эвристика, последовательно сокращающая число привлекаемых ТС. При заданном автопарке минимизация затрат на доставку грузов производится методами локального поиска [11, 14].

1. Постановка задачи

Рассмотрим полный взвешенный неориентированный граф $G = (V, A)$ с множествами вершин $V = \{0, 1, 2, \ldots, n\}$ и рёбер A. Вершина 0 будет соответствовать складу, где находятся грузы и ТС. Остальные вершины представляют клиентов, $V' = V \setminus \{0\}$. Для каждой пары вершин $(i, j) \in A$ известно расстояние между ними d_{ij} и время проезда t_{ij}. Для каждого клиента $i \in V'$ определена длительность обслуживания τ_i и требование на доставку q_i единиц груза. Обслуживание клиента i должно начинаться внутри временно́го окна $[e_i, l_i]$.

Автопарк состоит из различных ТС, их множество обозначим через K. Для каждого $k \in K$ задана грузоподъёмность Q_k, удельные затраты c_k на перевозку грузов, единовременные фиксированные затраты f_k, связанные с использованием ТС, и рабочая смена $u(k)$. Пусть $U = \{1, \ldots, m\}$ — множество рабочих смен. Каждая смена u имеет временно́е окно $[E_u, L_u]$, в котором должен начинаться и заканчиваться маршрут. Кроме того, смена u должна включать в себя t_u перерывов $P_u = \left\{p_u^1, \ldots, p_u^{t_u}\right\}$. Каждый перерыв имеет свою длительность и временно́е окно. Перерывы не могут идти подряд, между ними должен обслуживаться хотя бы один клиент. Если в маршруте мало клиентов, то число перерывов должно быть сокращено.

Маршрут каждого ТС начинается и заканчивается на складе. Стоимость маршрута складывается из фиксированной стоимости для привлечения ТС и удельной стоимости перевозки, умноженной на длину маршрута. Задача состоит в нахождении такого набора маршрутов и назначения им ТС, чтобы каждый клиент посещался ровно один раз, число привлекаемых ТС не превышало численности автопарка, загрузка каждого ТС не превышала его вместимости, а суммарная стоимость маршрутов была минимальной. Аналогичная задача без временны́х окон и перерывов рассматривалась в [1].

2. Математическая модель

Представим задачу в терминах частично-целочисленного программирования. Для каждого клиента $i \in V'$ введём $\sum\limits_{u \in U} t_u$ фиктивных клиентов. Обслуживание такого клиента будет требовать перерыва в работе водителя. Перерыв следует сразу после обслуживания клиента, либо через некоторое время до посещения следующего клиента в маршруте.

Пусть P — список всех перерывов, $P = \big(P_1^1, \ldots, P_1^{t_1}, \ldots, P_m^1, \ldots, P_m^{t_m}\big)$. Через $i\big(P_u^j\big)$ обозначим позицию перерыва P_u^j в списке P. Определим новый набор индексов

$$W = V \cup V_1 \cup \cdots \cup V_{|P|} \cup \{0'\},$$

где $V_i = \{in + i, \ldots, (i+1)n + i\}$ — множество вершин, связанных с перерывом $i \in P$. Последняя вершина $0'$ с номером $|W|$ обозначает склад как конечный пункт любого маршрута. Через $V(u) = \{V_{i(P_u^1)}, \ldots, V_{i(P_u^{t_u})}\}$ обозначим подмножество вершин с перерывами для заданной рабочей смены u.

Введём расширенную матрицу $\{d'_{ij}\}$, $i, j \in W$, следующим образом:

$$d'_{ij} = \begin{bmatrix} \big[A_0^0\big] & \big[A_0^1\big] & \big[A_0^2\big] & \ldots & \big[A_0^{|P|}\big] & \big[A_0^{|P|+1}\big] \\ \big[A_1^0\big] & \big[A_1^1\big] & \big[A_1^2\big] & \ldots & \big[A_1^{|P|}\big] & \big[A_1^{|P|+1}\big] \\ \big[A_2^0\big] & \big[A_2^1\big] & \big[A_2^2\big] & \ldots & \big[A_2^{|P|}\big] & \big[A_2^{|P|+1}\big] \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \big[A_{|P|}^0\big] & \big[A_{|P|}^1\big] & \big[A_{|P|}^2\big] & \ldots & \big[A_{|P|}^{|P|}\big] & \big[A_{|P|}^{|P|+1}\big] \\ \big[A_{|P|+1}^0\big] & \big[A_{|P|+1}^1\big] & \big[A_{|P|+1}^2\big] & \ldots & \big[A_{|P|+1}^{|P|}\big] & \big[A_{|P|+1}^{|P|+1}\big] \end{bmatrix}.$$

Подматрица A_i^j имеет множество строк V_i и множество столбцов V_j. Подматрица A_0^0 совпадает с матрицей $\{d_{ij}\}$. Подматрицы A_0^k, $k = 1, \ldots, |P|$,

определим следующим образом:

$$d'_{ij} = \begin{cases} 0, & \text{если } j = i + kn + k, \\ \infty & \text{в противном случае,} \end{cases} \qquad i \in V,\ j \in V_1 \cup \cdots \cup V_{|P|}.$$

Вершина $j = i+kn+k$ соответствует перерыву в одной из смен. Полагая $d'_{ij} = \infty$ для $j \neq i + kn + k$, позволяем после клиента i сделать не более одного перерыва в одной из смен. При этом фактическое расстояние, преодолеваемое ТС от клиента i до этого перерыва, будет определяться позже из других соображений. Подматрицы $A^0_k,\ k = 1, \ldots, |P|$, определим следующим образом:

$$d'_{ij} = \begin{cases} \infty, & \text{если } j = i - kn - k, \\ d_{i-kn-k,j} & \text{в противном случае,} \end{cases} \qquad i \in V_1 \cup \cdots \cup V_{|P|},\ j \in V.$$

Запрет $d'_{ij} = \infty$ для $j = i - kn - k$ не даёт возможности посещать клиента i дважды — до и после перерыва. Элементы подматрицы $A^j_i,\ i, j, = 1, \ldots, |P|$, положим равными ∞, чтобы запретить два перерыва подряд. Подматрицы $A^{|P|+1}_k$ определяют расстояние до конечной вершины — склада $0'$, поэтому $d_{i+kn+k,|W|} = d_{i0}$ для $i \in V',\ k = 0, 1, \ldots, |P|$. Последняя строка подматриц определяет расстояние от последней вершины — склада — до других вершин. Поскольку склад является концом любого маршрута, положим $d'_{|W|j} = \infty,\ j \in W$.

Для каждого фиктивного клиента определим его временно́е окно как окно соответствующего перерыва. Длительность обслуживания такого клиента будет равна длительности перерыва. Введём переменные задачи:

$x_{ijk} = 1$, если k-е ТС едет из i в j, и $x_{ijk} = 0$ в противном случае,

$t_{ijk} \geqslant 0$ — время проезда из i в j для k-го ТС,

$s_{ik} \geqslant 0$ — начало обслуживания k-м ТС клиента i,

$y_{ik} = 1$, если k-е ТС посещает клиента i, и $y_{ik} = 0$ в противном случае.

Тогда задача записывается следующим образом: найти

$$\min \sum_{k \in K} f_k y_{0k} + \sum_{k \in K} c_k \sum_{i,j \in W} d'_{ij} x_{ijk} \tag{1}$$

при условиях

$$\sum_{i \in V'} q_i y_{ik} \leqslant Q_k y_{0k}, \quad k \in K, \tag{2}$$

$$\sum_{k \in K} y_{ik} = 1, \quad i \in V', \tag{3}$$

$$\sum_{i\in V(u(k))} y_{ik} \geqslant \min\Big\{t_{u(k)}, \sum_{j\in V'} y_{jk} + 1\Big\}, \quad k \in K, \tag{4}$$

$$\sum_{i\in V_l} y_{ik} \leqslant 1, \quad l \in P,\ k \in K, \tag{5}$$

$$n y_{0k} \geqslant \sum_{i\in V'} y_{ik}, \quad k \in K, \tag{6}$$

$$\sum_{i\in V\cup V(u(k))} x_{0ik} = y_{0k}, \quad k \in K, \tag{7}$$

$$y_{|W|k} = y_{0k}, \quad k \in K, \tag{8}$$

$$\sum_{i\in V\cup V(u(k))} x_{i|W|k} = y_{|W|k}, \quad k \in K, \tag{9}$$

$$\sum_{j\in W} x_{ijk} + \sum_{j\in W} x_{jik} = 2y_{ik}, \quad k \in K,\ i \in V' \cup V(u(k)), \tag{10}$$

$$\sum_{i\in W} x_{ijk} \leqslant 1, \quad j \in W,\ k \in K, \tag{11}$$

$$\sum_{i\in W} x_{jik} \leqslant 1, \quad i \in W,\ k \in K, \tag{12}$$

$$t_{ijk} = t_{ij}, \quad i,j \in V,\ k \in K, \tag{13}$$

$$t_{0ik} = t_{0i}, \quad i \in V,\ k \in K, \tag{14}$$

$$t_{i|W|k} = t_{i0}, \quad i \in V,\ k \in K, \tag{15}$$

$$t_{ijk} + t_{jlk} = t_{il}, \quad i,l \in V,\ j \in V(u(k)),\ k \in K, \tag{16}$$

$$t_{0jk} + t_{j,i,k} = t_{0i}, \quad i \in V,\ j \in V(u(k)),\ k \in K, \tag{17}$$

$$t_{ijk} + t_{j|W|k} = t_{i0}, \quad i \in V,\ j \in V(u(k)),\ k \in K, \tag{18}$$

$$s_{ik} + \tau_i + t_{ijk} - M(1 - x_{ijk}) \leqslant s_{jk}, \quad i,j \in W,\ k \in K, \tag{19}$$

$$e_i \leqslant s_{ik} \leqslant l_i, \quad i \in W,\ k \in K, \tag{20}$$

$$s_{0k} \geqslant E_{u(k)}, \quad k \in K, \tag{21}$$

$$s_{|W|k} \leqslant L_{u(k)}, \quad k \in K, \tag{22}$$

$$x_{ijk}, y_{ik} \in \{0,1\}, \quad t_{ijk} \geqslant 0, \quad s_{ik} \geqslant 0, \quad i,j \in W,\ k \in K. \tag{23}$$

Целевая функция (1) задаёт суммарные затраты на привлечение ТС и обслуживание клиентов. Неравенство (2) обеспечивает вместимость грузов в ТС. Равенства (3) гарантируют обслуживание каждого клиента одним ТС. Условие (4) определяет число перерывов в маршруте каждого ТС. Левая часть неравенства (4) либо равна числу перерывов в смене $u(k)$, либо меньше этой величины, если в маршруте мало клиен-

тов. Неравенство (5) позволяет каждый перерыв делать не более одного раза. Условие (6) требует от ТС выходить со склада, если ему предписано обслуживать клиентов. Равенство (7) определяет первого клиента в маршруте каждого ТС. Равенство (8) требует возвращения ТС на склад, если оно обслужило клиентов. Условие (9) определяет последнего клиента в маршруте каждого ТС. Условия (10)–(12) требуют приехать к клиенту и уехать от него, если он обслуживается k-м ТС. Остальные ограничения определяют расписание ТС, т. е. переменные s_{ik}, t_{ijk} и их связь с переменными x_{ijk}. Равенства (13)–(15) определяют время проезда между нефиктивными клиентами и складом. Равенства (16)–(18) позволяют делать перерыв во время переезда от одного клиента к другому или между складом и клиентом. Неравенство (19) исключает подциклы, не проходящие через склад. Здесь константа M выбирается достаточно большой, чтобы при $x_{ijk} = 0$ гарантировать выполнение неравенства при любых неотрицательных s_{jk}. Оставшиеся неравенства связаны с временны́ми окнами для обслуживания клиентов, рабочих смен и перерывов в них.

Представленная модель может быть переписана в терминах частично-целочисленного линейного программирования. Для этого достаточно линеаризовать условие (4). Введём вспомогательные переменные $p_k \geqslant 0$, $k \in K$, которые будут задавать правую часть неравенства (4) и булевы переменные z_k, $k \in K$, определяющие, на каком именно из двух элементов в (4) достигается минимум. Тогда условие (4) эквивалентно системе неравенств

$$\sum_{i \in V(u(k))} y_{ik} \geqslant p_k, \quad k \in K,$$

$$t_{u(k)} - Mz_k \leqslant p_k \leqslant t_{u(k)}, \quad k \in K,$$

$$\sum_{j \in V'} y_{jk} + 1 - M(1 - z_k) \leqslant p_k \leqslant \sum_{j \in V'} y_{jk} + 1, \quad k \in K.$$

Полученная модель позволяет использовать классические методы целочисленного линейного программирования, в частности, такие решатели как CPLEX, GUROBI и др. К сожалению, большое число переменных и ограничений не позволяет получать точное решение задачи даже при небольшом числе клиентов. В связи с этим ниже предлагается приближённый метод, основанный на идеях локального поиска.

3. Характеристики решений и окрестности

Пусть r — некоторый маршрут, содержащий n_r клиентов. Будем задавать его последовательностью $\sigma = \left(\sigma_0^r, \sigma_1^r, \ldots, \sigma_{n_r+1}^r\right)$, в которой пер-

вый и последний элементы соответствуют складу: $\sigma_0^r = \sigma_{n_r+1}^r = 0$. По заданной последовательности легко посчитать загрузку транспортного средства при выходе со склада $Q(r) = \sum\limits_{i=1,\ldots,n_r} q_{\sigma_i^r}$ и длину маршрута $C(r) = \sum\limits_{i=0,\ldots,n_r} d_{\sigma_i^r \sigma_{i+1}^r}$. Чтобы вычислить суммарное время ТС для маршрута r, потребуется такое понятие как *временно́е смещение* (time warp). Оно связано с временны́ми окнами клиентов. Если ТС прибывает к клиенту i раньше временно́го окна, $s_{ik} < e_i$, то возникает простой на величину $e_i - s_{ik}$. Если же ТС опаздывает к клиенту i, т. е. приходит позже временно́го окна, $s_{ik} > l_i$, то будем считать, что опоздания не было, время прибытия совпадает с окончанием временно́го окна: $s_{ik} = l_i$, но появилось временно́е смещение $tw_{i-1,i}$, равное величине опоздания. Тогда общее временно́е смещение для маршрута r определяется как сумма смещений: $TW(r) = \sum\limits_{i=1,\ldots,n_r} tw_{i-1,i}$. Длительность маршрута $D(r)$ с учётом временно́го смещения получается равной $D(r) = s_{\sigma_{n_r+1}k} - s_{0k} + TW(r)$.

Простой не нарушает допустимости решения. Временно́е смещение, наоборот, соответствует нарушению условий (20). В ходе локального поиска будет допускаться нарушение ограничений по вместимости ТС, длительности маршрутов и временны́м окнам клиентов. За каждое такое нарушение в целевую функцию будет вноситься штраф.

Пусть r и r' — два маршрута, задаваемые последовательностями σ^r и $\sigma^{r'}$ соответственно. Определим следующие три окрестности.

N_1 (Swap and Relocate). Две подпоследовательности $\sigma_i^r, \ldots, \sigma_j^r$ и $\sigma_{i'}^{r'}, \ldots, \sigma_{j'}^{r'}$ меняются одна на другую в своих маршрутах. Длины подпоследовательностей не превышают двух. Допускается нулевая подпоследовательность и совпадение маршрутов. В последнем случае подпоследовательности не должны пересекаться.

N_2 (2-opt*). Две подпоследовательности $\sigma_i^r, \ldots, \sigma_j^r$ и $\sigma_{i'}^{r'}, \ldots, \sigma_{j'}^{r'}$, содержащие крайние элементы в различных маршрутах, меняются одна на другую.

N_3 (2-opt). В маршруте r выбирается подпоследовательность $\sigma_i^r, \ldots, \sigma_j^r$ и порядок элементов в ней меняется на противоположный.

Каждая из окрестностей имеет мощность $O(n^2)$. Чтобы исключить из рассмотрения неперспективные решения, вводится специальная мера $\delta(v_i, v_j)$, показывающая близость вершин v_i, $v_j \in V'$ как по расстоянию, так и по временны́м окнам:

$$\delta(v_i, v_j) = d_{ij} + \delta^{WT} \max\{l_j - \tau_i - t_{ij} - l_i, 0\} + \delta^{TW} \max\{l_i + \tau_i + t_{ij} - l_j, 0\},$$

где весовые коэффициенты δ^{WT} и δ^{TW} определяют важность простоя

и временно́го смещения. Эмпирическим путём найдены наилучшие значения этих коэффициентов с точки зрения производительности локального поиска [14]: $\delta^{WT} = 0.2$, $\delta^{TW} = 1$. Пользуясь этим определением меры близости клиентов, можно сократить размер каждой окрестности. Для клиента v_i найдём Γ ближайших клиентов, их множество обозначим через $\Gamma(v_i)$. Тогда в окрестностях N_1 и N_2 будем просматривать только такие подпоследовательности $\sigma_i^r, \ldots, \sigma_j^r$ и $\sigma_{i'}^{r'}, \ldots, \sigma_{j'}^{r'}$, что $\sigma_i^r \in \Gamma\big(\sigma_{i'-1}^{r'}\big)$ или $\sigma_{i'}^{r'} \in \Gamma\big(\sigma_{i-1}^{r}\big)$. В результате мощность каждой из окрестностей сократится до $O(\Gamma n)$. Дальнейшее сокращение трудоёмкости будет проводиться путём рандомизации окрестностей. Наилучший переход будет выбираться после того, как просмотрено только 5 % соседних решений, выбранных случайным образом с равномерным распределением [1].

Переход по любой из введённых окрестностей можно представить как разбиение маршрутов на подпоследовательности и конкатенация их в новые маршруты. Операцию конкатенации будем обозначать символом $\oplus$. Для последовательности σ будем хранить минимальную длительность $D(\sigma)$, минимальное временно́е смещение $TW(\sigma)$, наиболее раннее $E(\sigma)$ и наиболее позднее $L(\sigma)$ время переезда к первой вершине в последовательности, допустимые для расписания с минимальной длительностью. Кроме того, будем хранить суммарное расстояние $C(\sigma)$ и суммарную загрузку $Q(\sigma)$. Аналогичные данные потребуются для всех подпоследовательностей в маршруте. Для подпоследовательности σ^0, состоящей только из одного клиента v_i, эти величины определяются следующим образом: $D(\sigma^0) = \tau_i$, $TW(\sigma^0) = 0$, $E(\sigma^0) = l_i$, $L(\sigma^0) = l_i$, $C(\sigma^0) = 0$, $Q(\sigma^0) = q_i$. Для последовательности из двух и более клиентов эти значения вычисляются по следующему правилу.

Утверждение 1 [14]. *Пусть $\sigma = (\sigma_i, \ldots, \sigma_j)$ и $\sigma' = (\sigma'_{i'}, \ldots, \sigma'_{j'})$ — две подпоследовательности. Значения характеристик для их конкатенации вычисляются по следующим формулам*:

$$
\begin{gathered}
D(\sigma \oplus \sigma') = D(\sigma) + D(\sigma') + t_{\sigma_j \sigma'_{i'}} + \Delta_{WT}, \\
TW(\sigma \oplus \sigma') = TW(\sigma) + TW(\sigma') + \Delta_{TW}, \\
E(\sigma \oplus \sigma') = \max\{E(\sigma') - \Delta, E(\sigma)\} - \Delta_{WT}, \\
L(\sigma \oplus \sigma') = \min\{L(\sigma') - \Delta, L(\sigma)\} + \Delta_{TW}, \\
C(\sigma \oplus \sigma') = C(\sigma) + C(\sigma') + d_{\sigma_j \sigma'_{i'}}, \\
Q(\sigma \oplus \sigma') = Q(\sigma) + Q(\sigma'),
\end{gathered}
$$

где $\Delta = D(\sigma) - TW(\sigma) + t_{\sigma_j \sigma'_{i'}}$, $\Delta_{WT} = \max\{E(\sigma') - \Delta - L(\sigma), 0\}$, $\Delta_{TW} = \max\{E(\sigma) + \Delta - L(\sigma'), 0\}$.

Таким образом для каждого соседнего решения его характеристики можно вычислить с трудоёмкостью $O(1)$ без учёта перерывов.

В случае с одним перерывом любая последовательность может быть охарактеризована двумя наборами данных — старым $D(\sigma)$, $TW(\sigma)$, $E(\sigma)$ и $L(\sigma)$ набором без перерыва и новым $\overline{D}(\sigma)$, $\overline{TW}(\sigma)$, $\overline{E}(\sigma)$ и $\overline{L}(\sigma)$ набором с одним перерывом. Для подпоследовательности σ^0 с одним клиентом v_i получаем $\overline{D}(\sigma^0) = \tau_i$, $\overline{TW}(\sigma^0) = \infty$, $\overline{E}(\sigma^0) = \infty$, $\overline{L}(\sigma^0) = 0$.

Обозначим через σ_p последовательность σ с одним перерывом v_p. Для конкатенации двух последовательностей σ и σ' с одним перерывом надо выбрать лучший из трёх вариантов:

(i) перерыв находится в σ,

(ii) перерыв находится в σ',

(iii) перерыв находится между σ и σ'.

Лучший вариант из множества $\Sigma = \{\sigma_p \oplus \sigma', \sigma \oplus \sigma'_p, \sigma \oplus v_p \oplus \sigma'\}$ должен иметь минимальную длительность, т. е.

$$(\sigma \oplus \sigma')_p = \arg\min_{\sigma \in \Sigma} D(\sigma).$$

Первые два варианта рассчитываются по тем же формулам, что и раньше, но при наличии перерыва в последовательности берётся набор характеристик ($\overline{D}$, $\overline{TW}$, $\overline{E}$, $\overline{L}$). Рассмотрим вариант (iii), для которого надо знать время проезда до перерыва.

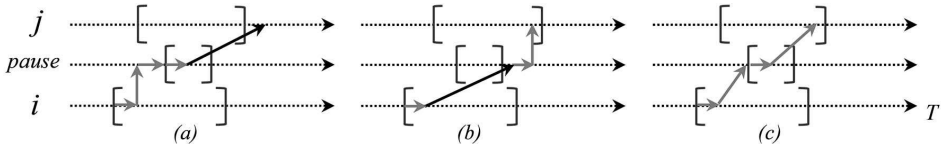

Рис. 1. Случаи вставки перерыва между клиентами

На рис. 1 показаны три варианта вставки перерыва между клиентами. В случае (a) вставляем перерыв непосредственно после обслуживания первого клиента. При такой вставке расстояние от клиента i до j преодолевается после перерыва, при этом необходимо дождаться начала временно́го окна для перерыва. В случае (b) начинаем перерыв сразу перед обслуживанием второго клиента. После затраченного времени на проезд до второго клиента становится невозможным выполнить требования по временно́му окну перерыва. В случае (c) часть времени на проезд потрачено до перерыва и часть времени — после. Это позволило удовлетворить все временны́е ограничения и минимизировать длительность обслуживания всей последовательности.

Теорема 1. *Оптимальное время до перерыва определяется формулой*

$$t_{\sigma v_p} = \min\{\max\{e_{v_p} - D(\sigma) + TW(\sigma) - L(\sigma), 0\}, t_{\sigma_j \sigma'_{i'}}\}.$$

Доказательство. Время проезда от последнего клиента σ до перерыва v_p выбирается так, что ожидание перерыва Δ_{WT} минимально. Время ожидания будет максимальным, если время проезда до перерыва равно нулю. В этом случае сразу после последнего клиента в σ в момент времени $D(\sigma) - TW(\sigma) + L(\sigma)$ ТС простаивает до момента e_{v_p}. Простой равен $\max\big\{0, e_{v_p} - \big(D(\sigma) - TW(\sigma) + L(\sigma)\big)\big\}$. Если эта величина не превышает времени переезда до первого клиента в σ', т. е. $t_{\sigma_j \sigma'_i}$, то время ожидания можно сделать нулевым, позволив ТС двигаться от σ_j к σ'_i и сделав перерыв через $t_{\sigma v_p} = \max\{e_{v_p} - D(\sigma) + TW(\sigma) - L(\sigma), 0\}$. В противном случае перерыв придётся делать непосредственно перед обслуживанием клиента $t_{\sigma'_i}$ через $t_{\sigma_j \sigma'_i}$ единиц времени. Таким образом, получаем

$$t_{\sigma v_p} = \min\{\max\{e_{v_p} - D(\sigma) + TW(\sigma) - L(\sigma), 0\}, t_{\sigma_j \sigma'_{i'}}\}.$$

Теорема 1 доказана.

Пользуясь полученным выражением, можно найти оптимальное размещение перерывов $\big\{p_u^1, \ldots, p_u^{t_u}\big\}$ для смены u в последовательности σ, если число клиентов в ней не меньше числа перерывов t_u. Для этого достаточно рассмотреть все варианты $C_{|\sigma|+1}^{t_u}$ размещения перерывов и выбрать наилучший. Если такой подход оказывается трудоёмким, то можно применить технику динамического программирования для приближённого решения задачи.

Пусть величины $\overline{D}(\sigma, u, i, j)$, $\overline{TW}(\sigma, u, i, j)$, $\overline{E}(\sigma, u, i, j)$, $\overline{L}(\sigma, u, i, j)$ задают значения характеристик последовательности σ, содержащей перерывы $\big\{p_u^i, \ldots, p_u^j\big\}$, $1 \leqslant i \leqslant j \leqslant t_u$, для смены u. Такую последовательность далее будем обозначать через σ_{uij}. Для конкатенации $(\sigma \oplus \sigma')$ расстановку перерывов будем выбирать из следующего множества:

$$\begin{aligned}\Sigma_{uij} = {} & \{\sigma \oplus \sigma'_{uij}\} \cup \{\sigma_{uik} \oplus \sigma'_{u,k+1,j} \mid i \leqslant k < j\} \cup \{\sigma_{uij} \oplus \sigma'\} \\ & \cup \big\{\sigma \oplus P_u^i \oplus \sigma'_{u,i+1,j}\big\} \cup \big\{\sigma_{u,i,k-1} \oplus P_u^k \oplus \sigma'_{u,k+1,j} \mid i \leqslant k \leqslant j\big\} \\ & \cup \big\{\sigma_{u,i,j-1} \oplus P_u^j \oplus \sigma'\big\}.\end{aligned}$$

Положим

$$(\sigma \oplus \sigma')_{uij} = \arg \min_{\sigma \in \sum_{uij}} D(\sigma).$$

Таким образом, для просмотра окрестностей требуется вычислить все значения $\overline{D}(\sigma, u, i, j)$, $\overline{TW}(\sigma, u, i, j)$, $\overline{E}(\sigma, u, i, j)$, $\overline{L}(\sigma, u, i, j)$ для всех $i, j \in 1, \ldots, t_n$, $u \in U$ и каждой последовательности σ в маршруте.

4. Алгоритм решения задачи

Общая схема алгоритма состоит из трёх последовательных шагов. На первом шаге делается попытка найти допустимое решение задачи. Заметим, что проверка совместимости системы ограничений (2)–(23) является NP-трудной задачей даже при отсутствии временны́х окон у клиентов, рабочих смен и перерывов. Поэтому уже на первом шаге алгоритм может закончить работу. Если же допустимое решение получено, то на втором шаге минимизируется число используемых ТС. Предполагается, что величины $f_k,\ k \in K$, определяющие стоимость привлечения ТС, существенно больше транспортных издержек. На третьем шаге при заданном автопарке минимизируются транспортные издержки для обслуживания всех клиентов.

4.1. Построение допустимого решения. Маршрут ТС называют *допустимым*, если он удовлетворяет ограничениям по временны́м окнам, содержит требуемое число перерывов и загрузка ТС не превышает предельной загрузки. В противном случае маршрут называют *недопустимым*. Допустимое решение задачи состоит из набора допустимых маршрутов, причём каждый клиент посещается только одним ТС. Под *частичным решением* будем понимать набор допустимых маршрутов, позволяющий обслужить некоторое подмножество клиентов, но каждый клиент из этого подмножества посещается только одним ТС.

Алгоритм построения допустимого решения будет состоять из двух этапов. На первом этапе строится частичное решение задачи и выделяется множество C необслуженных клиентов. На втором этапе делается попытка вставить необслуженных клиентов в маршруты, возможно, с нарушением ограничений по вместимости ТС и временны́м окнам, а затем сократить эти нарушения до нуля, используя штрафные функции и локальный поиск.

Определим функцию штрафа $F_{\text{penalty}}(r, k, u)$ для k-го ТС, работающего в смену u на маршруте r, как сумму штрафов за превышение длительности рабочей смены, вместимости ТС и наличия временны́х смещений:

$$F_{\text{penalty}}(r, k, u) = c_d \max\{0, D(r) - L_u + E_u\} + c_Q \max\{0, Q(r) - Q_k\} + c_{tw} TW(r).$$

Значения штрафных коэффициентов c_d, c_Q, c_{tw} подбираются в ходе локального поиска (см. п. 4.3). Функция штрафа $F_{\text{penalty}}(s)$ для решения s определяется как сумма штрафных функций по всем маршрутам.

На первом этапе применяется жадный алгоритм с последовательным заполнением ТС [1]. При вставке очередного клиента проверяются условия по временны́м окнам и возможности для перерывов в течение смены. На втором этапе для полученного частичного решения s и множества C необслуженных клиентов применяется схема, впервые предложенная в [12]. Псевдокод этой процедуры представлен в алгоритме 1.

Algorithm 1: Вставка клиентов

```
1  Procedure AddCustomers(C, s, T^1_max);
2  begin
3      EP ⟵ C ; // создали очередь необслуженных клиентов
4      while EP ≠ ∅ ∧ time < T^1_max do
5          Взять первого клиента v_in из очереди EP;
6          if N^f_insert(v_in, s) ≠ ∅ then
7              Выбрать решение s' ∈ N^f_insert(v_in, s) случайным образом;
8          else
9              Вставить v_in в решение s с минимальным штрафом;
10             s' ⟵ RVND(s, F_penalty);
11             if F_penalty(s') ≠ 0 then
12                 s' ⟵ s ; // получили недопустимое решение
13         end
14         s ⟵ s';
15         if F_penalty(s) ≠ 0 then
16             V_out ⟵ EjectCustomers(s);
17             Добавить множество V_out в конец очереди EP;
18             s ⟵ Perturb(s) ; // перестроили решение
19         end
20     end
21     if EP ≠ ∅ then
22         Восстановить решение s в начальном состоянии;
23     end
24     Предъявить решение s;
25 end
```

Алгоритм работает до тех пор, пока все клиенты не будут обслужены либо не закончится отведённое ему время работы $T^1_{\max}$. Множество C хранится в виде очереди. На каждой итерации основного цикла (строки 4–21) из очереди извлекается первый элемент v_{in} и делается попытка

его вставки в частичное решение (строки 6–14). Если это удаётся сделать, т. е. множество $N_{\text{insert}}(v_{\text{in}}, s)$ частичных решений, получающихся вставкой этого клиента, непусто, то случайным образом с равномерным распределением выбирается одно из таких решений и запоминается как текущее решение (строки 6–8). В противном случае клиент v_{in} вставляется с минимальным штрафом и применяется процедура локального поиска RVND в надежде найти допустимое частичное решение (строки 10–11). Процедура RVND представляет собой спуск по чередующимся окрестностям и в данной случае минимизирует величину F_{penalty}. Подробнее она описана в п. 4.3. Если допустимое решение удалось найти, то оно сохраняется (строка 14). В противном случае из решения удаляется некоторое подмножество клиентов V_{out} для получения частичного решения (процедура EjectCustomers), вносятся случайные изменения (процедура Perturb) и клиенты из множества V_{out} добавляются в конец очереди (строки 18–19). Если по окончании основного цикла очередь так и не оказалась пустой, то решение s возвращается в начальное состояние (строка 23).

Algorithm 2: Удаление маршрутов

```
1  Procedure RouteMinimizationHeuristic(s, T^2_max);
2  begin
3      while time < T^2_max do
4          Выбрать маршрут r случайным образом и удалить его из s;
5          C ⟵ r;
6          AddCustomers(C, s, T^1_max);
7          if F_penalty(s) ≠ 0 then
8              Восстановить решение s в начальном состоянии;
9          end
10     end
11     Предъявить решение s;
12 end
```

Процедура EjectCustomers хранит статистику $q_{v_{\text{in}}}$ по числу неудачных вставок клиента v_{in} [12]. Величина $q_{v_{\text{in}}}$ обнуляется при удачной вставке и увеличивается на единицу при каждой неудачной вставке. Множество V_{out} выбирается таким, что $\sum\limits_{v \in V_{\text{out}}} q_v$ минимальна. Процедура Perturb выполняет заданное число случайных переходов по указанным выше окрестностям с сохранением допустимости решений.

4.2. Минимизация числа маршрутов. Алгоритм минимизации числа маршрутов работает по следующему принципу [6]. На каждом шаге алгоритма случайным образом выбирается одно из ТС и его маршрут удаляется из решения задачи. Клиенты этого маршрута добавляются в множество C необслуженных клиентов, и к полученному частичному допустимому решению применяется алгоритм 1. Если в результате его работы получено новое допустимое решение с меньшим числом маршрутов, то это решение запоминается. В противном случае попытка сократить число маршрутов считается неудачной, удалённый маршрут возвращается обратно и шаг алгоритма заканчивается. Шаги повторяются до тех пор, пока не сработает критерий остановки. В качестве такого критерия может выступать, например, время работы алгоритма $T^2_{\max}$. Его общая схема представлена в алгоритме 2.

Algorithm 3: RVND

```
1  Procedure RVND(s, F);
2  begin
3      Создать список окрестностей NL;
4      while NL ≠ ∅ do
5          Выбрать случайным образом окрестность N из NL;
6          Найти наилучшее решение s' в N;
7          if F(s') < F(s) then
8              s ⟵ IntraRouteSearch(s', F); // локальное улучшение
                   измененных маршрутов
9              Восстановить список NL;
10         else
11             Удалить окрестность N из списка NL;
12         end
13     end
14     Предъявить решение s;
15 end
```

4.3. Минимизация транспортных издержек. При фиксированном числе маршрутов и выбранных для них ТС дальнейшее сокращение транспортных издержек осуществляется методами локального поиска [1, 11]. Ключевой процедурой здесь будет алгоритм локального улучшения с чередующимися окрестностями RVND. На вход этого алгоритма подаётся решение s и минимизируемая функция F, которая совпадает либо со штрафной функцией F_{penalty}, либо с функцией $\overline{F}(s) =$

$F_1(s) + F_{\text{penalty}}(s, c_d, c_Q, c_{tw})$, где $F_1(s)$ — целевая функция (1) без первого слагаемого. При поиске допустимого решения задачи и минимизации числа маршрутов используется только штрафная функция. При минимизации транспортных издержек — обе функции. Псевдокод процедуры RVND представлен в алгоритме 3.

На первом шаге алгоритма 3 составляется список окрестностей NL для обмена клиентами между маршрутами (см. разд. 3). На каждой итерации основного цикла (строки 4–13) случайным образом выбирается окрестность из этого списка и в ней находится наилучшее соседнее решение (строка 6). Если найденное решение хуже текущего, то окрестность удаляется из списка. В противном случае делается попытка улучшить полученное решение с помощью процедуры IntraRouteSearch, а список NL возвращается в исходное состояние. Процедура IntraRouteSearch работает по тому же принципу случайного выбора окрестностей на каждой итерации. Каждый изменённый маршрут улучшается обособленно, как если бы мы улучшали маршрут коммивояжёра. Используются только окрестности, которые меняют порядок посещения клиентов внутри одного маршрута.

Algorithm 4: Educate

```
1  Procedure Educate(s);
2  begin
3      {c_d, c_q, c_tw} ⟵ {c_d^0, c_q^0, c_tw^0}; // начальные штрафные
       коэффициенты
4      while штрафные коэффициенты не достигнут максимума do
5          s ⟵ RVND(s, F̄(c_d, c_q, c_tw));
6          s ⟵ RVND(s, F_penalty(c_d, c_q, c_tw));
7          if F_penalty(s) = 0 then
8              Предъявить решение s;
9          else
10             Увеличить штрафные параметры c_d, c_q, c_tw;
11         end
12     end
13     Вернуть решение s в исходное состояние;
14 end
```

В ходе локального поиска приходится постоянно выходить за пределы допустимой области, а затем с помощью штрафной функции возвращаться обратно. Такой механизм поиска реализован в процедуре Edu-

cate. Выбор штрафных коэффициентов играет здесь важную роль. Сначала они полагаются достаточно малыми. В процессе работы локального поиска делаются многократные попытки получить допустимое решение задачи. Каждая неудачная попытка влечёт рост штрафных коэффициентов. Алгоритм 4 представляет общую схему этой процедуры.

Algorithm 5: Локальный поиск

```
1  Procedure LocalSearch(s, T^3_max);
2  begin
3      s* ⟵ s;
4      while time< T^3_max do
5          s ⟵ Educate(s);
6          if F_penalty(s) = 0 then
7              if F_1(s) < F_1(s*) then
8                  s* ⟵ s;
9              end
10         Perturb(s) ; // диверсификация поиска
11     end
12     Предъявить решение s*;
13 end
```

В строке 3 алгоритма 4 штрафным коэффициентам присваиваются начальные значения. В основном цикле алгоритма (строки 4–12) осуществляется локальное улучшение по функции $\overline{F}$. Затем применяется процедура RVND со штрафной функцией. Если удалось получить допустимое решение (строка 9), то алгоритм заканчивает свою работу. В противном случае переходим к следующей итерации с бóльшими значениями штрафных коэффициентов. Если коэффициенты достигли максимально допустимых значений, а допустимое решение задачи так и не найдено, то в качестве результата работы предъявляется стартовое решения.

Общая схема локального поиска представлена в алгоритме 5. На вход алгоритма подаётся некоторое допустимое решение задачи для фиксированного автопарка. На каждой итерации основного цикла (строки 4–11) вызывается процедура Educate, а затем процедура диверсификации Perturb. Наилучшее из найденных допустимых решений s^* предъявляется в качестве результата работы алгоритма.

5. Численные эксперименты

Разработанные алгоритмы запрограммированы на языке Java и тестировались на сервере с процессором Intel Xeon 3.07 ГГц. В качестве тестовых примеров использовались два набора данных реальных поставок продукции в Новосибирске. В первом примере все клиенты из центрального района, во втором примере клиенты распределены по разным районам города. В каждом примере 1000 клиентов. Временно́е окно у каждого клиента составляет два часа, время обслуживания — от 3 до 30 минут, но основная часть клиентов обслуживается от 3 до 5 минут. За это время клиент получает до 1275 единиц товара. Автопарк состоит из 30 ТС: 24 машины с грузоподъёмностью 2500, 4 машины с грузоподъёмностью 3000, 2 машины с грузоподъёмностью 5000. Начальные затраты у всех машин одинаковые, $f_k = 10^6,\ k \in K$, удельные затраты c_k равны единице. За каждой машиной закреплён водитель, который работает в одну из трёх смен: с 7:00 до 16:00, с 13:00 до 23:00, с 7:00 до 23:00. У каждой смены три перерыва по 40 минут.

Таблица 1

Время просмотра окрестности

Пример	t_{search}	t_{prep}	%
1/0	202	20	9,9
1/1	480	90	18,75
1/2	794	160	20,15
1/3	1015	263	25,91
2/0	192	23	11,98
2/1	299	50	16,72
2/2	543	107	19,70
2/3	694	183	26,36

В первом эксперименте исследовалось влияние числа перерывов в смене на трудоёмкость локального поиска в алгоритме RVND. Табл. 1 показывает среднее время просмотра окрестности Relocate при числе перерывов от 0 до 3. В первом столбце указаны номер примера и число перерывов. Во втором столбце — среднее время в миллисекундах на просмотр окрестности. В третьем и четвёртом столбцах указано время на предпроцессинг и его доля в процентах от среднего времени счёта. Напомним, что при просмотре окрестности требуется вычисление характеристик для всех подпоследовательностей, порождающих соседние решения. Из табл. 1 видно, что наличие перерывов заметно усложняет расчёты, а доля предпроцессинга растёт от 10–12% до 25–26%. Тем

не менее это существенно меньше полного перебора вставки перерывов в маршруты, так как каждый маршрут в среднем состоит из 40–50 клиентов.

Во втором эксперименте исследовались возможности сокращения автопарка и влияние перерывов на этот процесс. В табл. 2 представлены результаты тестирования по 30 испытаниям для каждого примера с заданным числом перерывов. Время работы алгоритма ограничивалось двумя часами, из которых 10 минут отводилось на поиск допустимого решения, 60 минут — на минимизацию числа ТС, 50 минут — на сокращение длины маршрутов. На удаление одного маршрута выделялось не более двух минут в алгоритме 2. Число шагов в процедуре диверсификации Perturb ограничивалось величиной 100. Штрафные коэффициенты в алгоритме 4 полагались равными 10 и умножались на 10 на каждой итерации. Максимальное значение этих коэффициентов ограничивалось величиной 1000.

Таблица 2

Результаты работы алгоритма

	Число ТС	Пробег					
N	min / max	$D^0_{\min}$	$D^0_{\max}$	$D_{\min}$	$D_{\max}$	D_{avg}	σ (%)
1/0	25/25	1007427	1137817	373722	385801	376743	3766 (1,00)
1/1	25/25	988574	1088719	388511	467417	408824	23208 (5,68)
1/2	25/25	1014496	1117109	412426	552966	498162	43194 (8,67)
1/3	25/25	1020285	1113830	482812	630404	542472	46021 (8,48)
2/0	17/18	1138816	1316069	876867	1114978	1008252	81300 (8,06)
2/1	18/19	1167424	1318621	886321	1160035	1006542	98025 (9,74)
2/2	20/21	1324955	1514604	927951	1169061	1044429	93869 (8,99)
2/3	20/21	1169061	1531777	980749	1234636	1093872	104534 (9,56)

В первом столбце табл. 2 указаны номер примера и число перерывов. Во втором столбце приводятся минимальное и максимальное число ТС в 30 испытаниях алгоритма после оптимизации автопарка. В третьем и четвёртом столбцах показано минимальное и максимальное значения суммарного расстояния перед локальным поиском ($D^0_{\min}, D^0_{\max}$). Следующие три столбца показывают результаты локального поиска как минимальное, максимальное и среднее значения суммарного пройденного расстояния ($D_{\min}, D_{\max}, D_{\text{avg}}$). Последний столбец таблицы показывает значение σ стандартного отклонения суммарного расстояния от средней величины D_{avg}. Интересно отметить, что в первом примере, где клиенты разбросаны по всему городу, наличие перерывов не приводит к росту

числа ТС. Для второго примера это не так, и число ТС растёт от 18 до 21, что связано со значительно бóльшими расстояниями между клиентами. В обоих случаях наличие перерывов заметно меняет структуру решения и приводит к росту транспортных издержек.

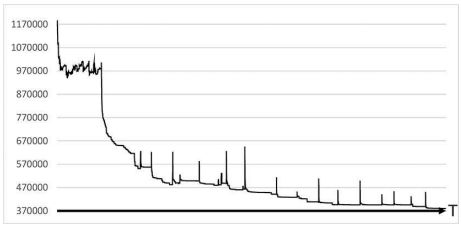

Рис. 2. Изменение транспортных издержек с ростом числа итераций

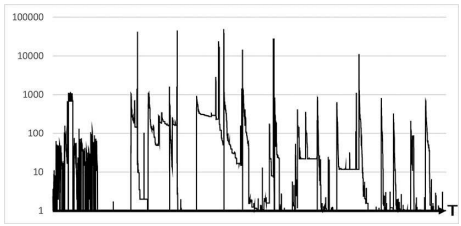

Рис. 3. Изменение величины штрафа с ростом числа итераций

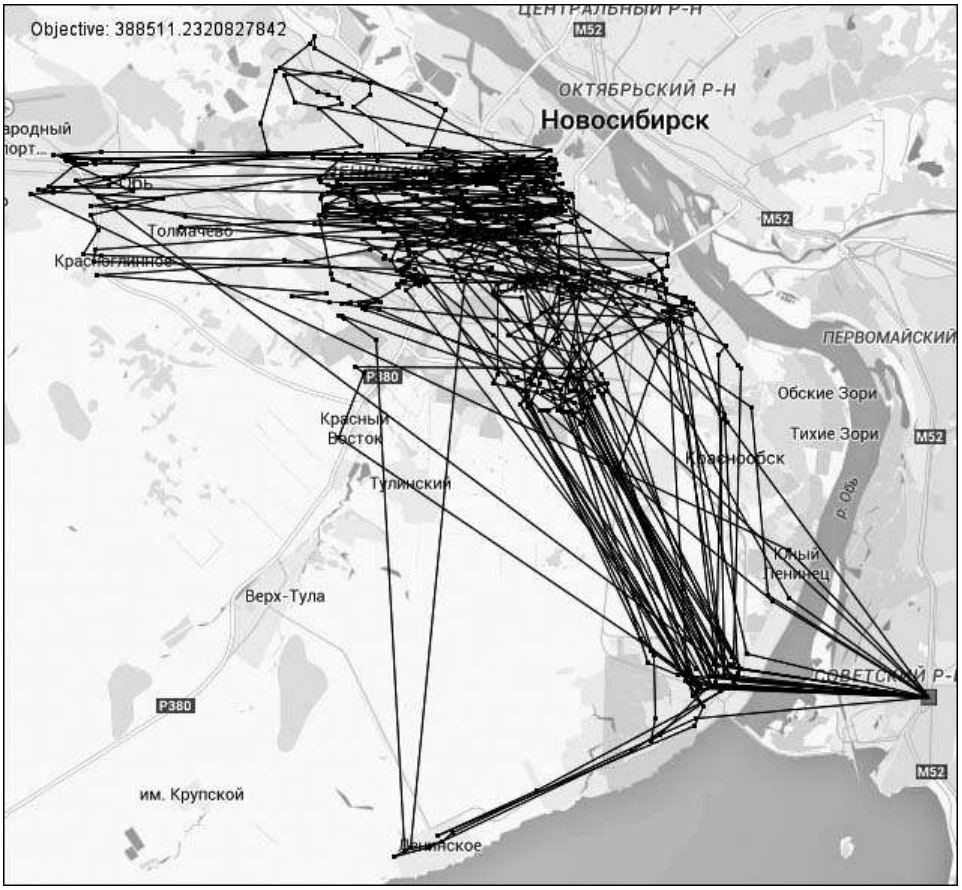

Рис. 4. Результат работы алгоритма для первого примера

На рис. 2 и 3 показаны изменения транспортных издержек и суммарного штрафа в ходе работы алгоритма. На рис. 4 представлен пример итогового решения с привязкой к карте г. Новосибирска.

Заключение

В работе представлена новая задача оптимизации разнородного автопарка, учитывающая наличие временны́х окон при обслуживании клиентов, рабочих смен водителей и перерывов в их работе. Построена математическая модель в терминах целочисленного линейного программирования. Разработан метод динамического программирования для расстановки перерывов в маршрутах. Этот метод используется для оценки соседних решений в ходе локального поиска и оптимизации автопарка. Проведены численные эксперименты на реальных исходных данных. Эксперименты показали высокую эффективность предложенного подхода и возможности его применения на практике.

ЛИТЕРАТУРА

1. **Кочетов Ю. А., Хмелёв А. В.** Гибридный алгоритм локального поиска для задачи маршрутизации транспортных средств с неоднородным автопарком // Дискрет. анализ и исслед. операций. 2015. Т. 22, № 5. С. 5–29.
2. **Bent R., van Hentenryck P.** A two-stage hybrid local search for the vehicle routing problem with time windows // Transp. Sci. 2004. Vol. 38, No. 4. P. 515–530.
3. **Bostel N., Dejax P., Guez P., Tricoire F.** Multiperiod planning and routing on a rolling horizon for field force optimization logistics // The vehicle routing problem: Latest advances and new challenges. Vol. 43. New York: Springer-Verl., 2008. P. 503–525.
4. **Bräysy O., Gendreau M.** Vehicle routing problem with time windows. Part I: Route constraction and local search algorithms // Transp. Sci. 2005. Vol. 39. P. 104–118.
5. **Bräysy O., Gendreau M.** Vehicle routing problem with time windows. Part II: Metaheuristics // Transp. Sci. 2005. Vol. 39, No. 1. P. 119–139.
6. **Gagliardi J.-Ph., Renaud J., Ruiz A., Coehlo C. L.** The vehicle routing problem with pauses // Tech. Rep. 2014–22, CIRRELT, Montreal, QC, Canada, 2014.
7. **Gehring H., Homberger J.** Parallelization of a two-phase metaheuristic for routing problems with time windows // Asia-Pac. J. Oper. Res. 2001. Vol. 18, No. 1. P. 35–47.
8. **Goel A.** Vehicle scheduling and routing with drivers' working hours // Transp. Sci. 2009. Vol. 43, No. 1. P. 17–26.

9. **Goel A., Archetti C., Savelsbergh M.** Truck driver scheduling in Australia // Comput. Oper. Res. 2012. Vol. 39, No. 5. P. 1122–1132.
10. **Goel A., Vidal T.** Hours of service regulations in road freight transport: An optimization-based international assessment // Transp. Sci. 2014. Vol. 48, No. 3. P. 391–412.
11. **Mladenović N., Hansen P.** Variable neighborhood search // Comput. Oper. Res. 1997. Vol. 24, No. 11. P. 1097–1100.
12. **Nagata Yu., Bräysy O.** A powerful route minimization heuristic for the vehicle routing problem with time windows // Oper. Res. Lett. 2009. Vol. 37, No. 5. P. 333–338.
13. **Sahoo S., Kim S., Kim B.-I., Kraas B., Popov A., Jr.** Routing optimization for waste management // Interfaces. 2005. Vol. 35, No. 1. P. 24–36.
14. **Vidal T., Crainic T. G., Gendreau M., Prins C.** A unified solution framework for multi-attribute vehicle routing problems // Eur. J. Oper. Res. 2014. Vol. 234, No. 3. P. 658–673.

Хмелёв Алексей Владимирович

Статья поступила
3 июня 2015 г.

Исправленный вариант —
11 августа 2015 г.

DISKRETNYI ANALIZ I ISSLEDOVANIE OPERATSII
November–December 2015. Volume 22, No. 6. P. 55–77

UDC 519.85 DOI: 10.17377/daio.2015.22.496

A THREE-PHASE HEURISTIC FOR THE VEHICLE FLEET AND ROUTE OPTIMIZATION

A. V. Khmelev[1]
[1] Novosibirsk State University,
2 Pirogov St., 630090 Novosibirsk, Russia
e-mail: avhmel@gmail.com

Abstract. We consider the applied vehicle routing problem with time windows. Each driver works in a given shift. Each shift has the start, finish and set of pauses which need to be scheduled en route. We introduce the mathematical formulation for this problem in terms of the mixed integer linear programming. In order to tackle large instances, we developed a three-phase local search algorithm with an effective procedure for search in neighbourhoods. Computational experiments for instances from one delivery company have shown the efficiency of the developed algorithm and significant reduction in costs. Tab. 2, ill. 4, bibliogr. 13.

Keywords: vehicle routing, time window, work shift, pause, local search, fleet optimization.

REFERENCES

1. **Yu. A. Kochetov** and **A. V. Khmelev,** Hybrid local search for the heterogenous fixed fleet vehicle routing problem, *Diskretn. Anal. Issled. Oper.,* **22**, No. 5, 5–29, 2015.
2. **N. Bent** and **P. van Hentenryck,** A two-stage hybrid local search for the vehicle routing problem with time windows, *Transp. Sci.,* **38**, No. 4, 515–530, 2004.
3. **N. Bostel, P. Dejax, P. Guez,** and **F. Tricoire,** Multiperiod planning and routing on a rolling horizon for field force optimization logistics, in B. Golden, S. Raghavan, and E. Wasil, eds., *The Vehicle Routing Problem: Latest Advances and New Challenges,* pp. 503–525, Springer, New York, 2008 (Oper. Res./Comput. Sci. Interfaces, Vol. 43).
4. **O. Bräysy** and **M. Gendreau,** Vehicle routing problem with time windows. Part I: Route construction and local search algorithms, *Transp. Sci.,* **39**, No. 1, 104–118, 2005.
5. **O. Bräysy** and **M. Gendreau,** Vehicle routing problem with time windows. Part II: Metaheuristics, *Transp. Sci.,* **39**, No. 1, 119–139, 2005.

6. **J.-Ph. Gagliardi, J. Renaud, A. Ruiz,** and **L. C. Coelho,** The vehicle routing problem with pauses, *Tech. Rep. CIRRELT-2104-22,* Interuniv. Res. Cent. Enterp. Netw., Logist. Transp., Montreal, Canada, 2014. Available at https://www.cirrelt.ca/DocumentsTravail/CIRRELT-FSA-2014-22.pdf. Accessed Oct. 15, 2015.
7. **H. Gehring** and **J. Homberger,** Parallelization of a two-phase metaheuristic for routing problems with time windows, *Asia-Pac. J. Oper. Res.,* **18**, No. 1, 35–47, 2001.
8. **A. Goel,** Vehicle scheduling and routing with drivers' working hours, *Transp. Sci.,* **43**, No. 1, 17–26, 2009.
9. **A. Goel, C. Archetti,** and **M. Savelsbergh,** Truck driver scheduling in Australia, *Comput. Oper. Res.,* **39**, No. 5, 1122–1132, 2012.
10. **A. Goel** and **T. Vidal,** Hours of service regulations in road freight transport: An optimization-based international assessment, *Transp. Sci.,* **48**, No. 3, 391–412, 2014.
11. **T. Mladenović** and **P. Hansen,** Variable neighborhood search, *Comput. Oper. Res.,* **24**, No. 11, 1097–1100, 1997.
12. **Yu. Nagata** and **O. Bräysy,** A powerful route minimization heuristic for the vehicle routing problem with time windows, *Oper. Res. Lett.,* **37**, No. 5, 333–338, 2009.
13. **S. Sahoo, S. Kim, B.-I. Kim, B. Kraas,** and **A. Popov, Jr.,** Routing optimization for waste management, *Interfaces,* **35**, No. 1, 24–36, 2005.
14. **T. Vidal, T. G. Crainic, M. Gendreau,** and **C. Prins,** A unified solution framework for multi-attribute vehicle routing problems, *Eur. J. Oper. Res.,* **234**, No. 3, 658–673, 2014.

Aleksey V. Khmelev

Received
3 June 2015
Revised
11 August 2015