“Исследование настроечных параметров алгоритма поиска с запретами и его модификация для решения курьерской задачи маршрутизации”

L. Lyubchik, R. Shafeyev

В статье рассматривается задача маршрутизации курьерской доставки с долгим сроком службы, для которой была построена дискретная модель и вычислительная схема, в основу которой был положен алгоритм поиска с запретами. Эффективность схемы предложенного алгоритма была проверена на задачах большой размерности, которые были составлены на базе известных модельных задачах маршрутизации на транспорте различных классов.

Introduction

Одним из средств экономии ресурсов при транспортировке грузов является применение систем поддержки принятия решений в сфере транспортной логистики. Одной из ключевых функций систем поддержки принятия решений в сфере транспортной логистики является возможность расчета и построения эффективных с точки зрения стоимости объезда маршрутов различного назначения на транспортной сети. Математическая формулировка этой задачи широко известно как задача маршрутизации транспорта. Существует ряд разновидностей ЗМТ с различными дополнительными условиями, которые позволяют учитывать грузоподъемность транспортных средств и другие ограничения для более полного представления деталей реальной действительности.

Данная статья посвящена рассмотрению задачи курьерской маршрутизации, которая является подзадачей ЗМТ. Данная задача состоит в нахождении маршрутов для посещения заданного множества адресов некоторым количеством единиц транспортных средств, выполняющих развозки товара от отправителя к получателю.

Автоматизация процесса планирования маршрутов будет актуальной для: интернет-магазинов, крупных оптовых компаний (например, для развозки товаров, которые быстро портятся или оптимизации схемы доставки товаров в пункты назначения), фирм с целью организации допоставок в магазины. Практическое применение задачи можно встретить в сфере транспортировки пациентов между корпусами поликлиники[16, 18], доставке газет и журналов, доставки топлива в частные дома [19]. Решение рассматриваемой задачи также имеет практическое применение в системах вертолетных перевозок людей между морскими нефтяными платформами [17].

В последнее время все больше людей пользуются услугами курьерских служб. Обычно заявки формируются днем ранее и у диспетчера имеется полный список заявок, которые необходимо выполнить в кратчайшие сроки.

Attanasio(2007) [15] провел тематическое исследование, в котором описал преимущества использования компьютерных методов перед человеческой диспетчеризацией. В своей работе он рассмотрел примеры работы eCourier Ltd, лондонской компании, которая предлагает курьерские услуги. Их клиентами в основном являются юридические фирмы, финансовые учреждения, рекламные агентства и другие организации, которые заинтересованы в быстрой доставке товаров или оригинальных подписях на документах. После того, как все заявки на конкретный момент времени сформированы, необходимо составить такой маршрут, который удовлетворял бы требованиям заказчиков. В зависимости от уровня обслуживания, который был задан клиентом, курьер может объединить товары нескольких клиентов в процессе развоза.

Компании, которые предлагают курьерские услуги достаточно часто имеют смешанный парк транспортных средств, который состоит из велосипедов, мотоциклов, автомобилей и небольших грузовиков. В зависимости от типа запроса, расположения транспортных средств и временных окон будет выбран тот вид транспортного средства, который сможет выполнить заказ в кратчайшие сроки. Исследования [15] показали, что использование компьютерных методов, в том числе алгоритмов оптимизации, были очень выгодными для курьерских компаний.

Таким образом, использование автоматизированной системы позволяет улучшить качество обслуживания, время доставки, повысить курьерскую эффективность и сократить расходы на доставку, тем самым обеспечить повышение конкурентного преимущества.

Постановка задачи

Пусть $C, dim(C) = n$ – множество транспортных средств, $Q, dim(Q) = m$ – множество запросов клиентов, которые требуется обработать на текущий момент времени.

Предположим, что следующая информация известна о транспортных средствах из множества C:
$\vec{P}_c$ – местоположение ТС, $c \in C$;
L_c – грузоподъемность ТС, $c \in C$.

Пусть S – множество отправителей груза, $(dim(S) = dim(Q) = m)$, R – множество получателей, $dim(R) = dim(Q) = m$. Тогда каждый клиентский запрос $q \in Q$ включает в себя следующую информацию:
s_q – отправитель груза клиента, $s \in S$;
r_q – получатель клиентского груза, $r \in R$;
$\vec{P}_{s_q}$ – Местоположение отправителя;
$\vec{P}_{r_q}$ – Местоположение получателя;
w_q – клиентский груз, который требуется перевезти от отправителя к получателю;
$[t_s^q, t_s^q + \Delta t_s^q]$ – временное окно, в течение которого нужно забрать груз у отправителя;

$[t_r^q, t_r^q + \Delta t_r^q]$ – временное окно, в течение которого нужно доставить груз получателю.

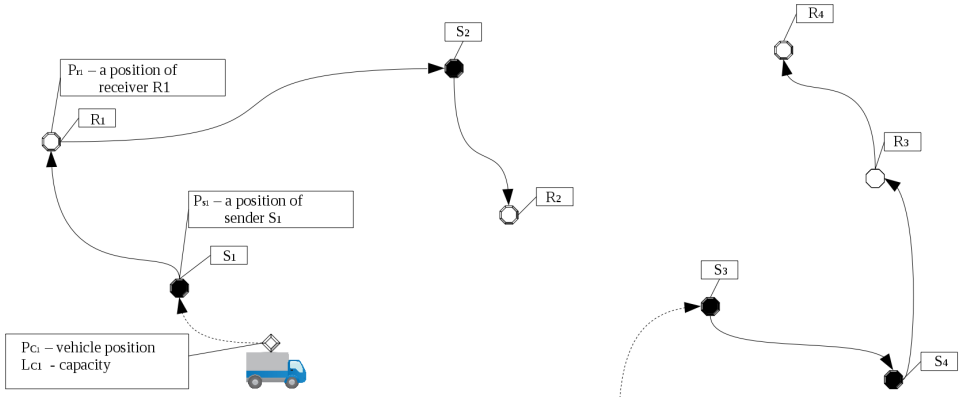
Таким образом, каждая заявка может быть представлена в виде кортежа:

$$\forall q \in Q : \exists q = (s_q, r_q, \vec{P}_{s_q}, \vec{P}_{r_q}, w_q, t_s^q, \Delta t_s^q, t_r^q, \Delta t_r^q) \tag{1}$$

Для того, чтобы оценить стоимость перевозки между пунктами назначения, определим функцию стоимости Ω:

$$\forall i, j \in S \cup R : \exists \Omega_{i,j} = \Omega(\vec{P}_i, \vec{P}_j) \tag{2}$$

Необходимо построить оптимальные маршруты движения транспортных средств для перевозки грузов от отправителя к получателю для всех клиентских запросов.



C_i	Vehicle
P_{C_i}	Vehicle position
S_q	Sender
P_{s_q}	Sender posinion
R_q	Receiver
P_{r_q}	Receiver position
W_q	Client shipment that is required to deliver from the sender

Рис. 1. Example of the Routing Delivery Problem.

Дискретная модель

Задачу маршрутизации можно представить в виде ориентированного графа $G = G(V, E)$. Пусть множество $V = C \cup S \cup R$ – узлы графа G, состоящее из

элементов множеств транспортных средств C, отправителей S и получателей R. E – динамическое множество дуг графа G, такое, что:

$$\forall e(X) \in E : e(X) = (v_i, v_j), \exists v_i \in V, v_j \in V/C \tag{3}$$

Пусть $X = \{X^k\}_{k=1}^n$ представляет собой последовательность матриц для каждого ТС $k \in C$. Элементы матрицы принимают следующие значения:

$$x_{i,j}^{(k)} = \begin{cases} 1, & \text{АТС } k \in C \text{ движется с узла } i \text{ в узел } j \\ 0, & \text{в противном случае.} \end{cases} \tag{4}$$

где: $i \in V/C \cup k, j \in V/C$.

Введем вектор переменных $\vec{Y}^k(X)$ для каждого транспортного средства $k\ inC$. Векторные элементы принимают следующие значения:

$$\vec{y_j^{(k)}}(X) = \begin{cases} 1, & \text{Запрос } j \in Q \text{ обслуживается ТС } k \in C \\ 0, & \text{в противном случае} \end{cases} \tag{5}$$

где: $i \in V/C \cup k, j \in V/C$.

Пусть $t_j^k(X^{(k)})$ – время прибытия автомобиля $k \in C$ в пункт назначения $j \in S \cup R$.

Целевая функция принимает следующий вид:

$$F(X) = \sum_{k \in C} \sum_{i,j \cup V} \Omega_{ij} \cdot x_{ij}^{(k)} \to min \tag{6}$$

Наложим ограничения на целевую функцию (6) для обеспечения непрерывности маршрутов:

$$\sum_{k \in C} \sum_{j \in S \cup R} x_{i,j}^{(k)} \leq 1, \forall i \in V \tag{7}$$

$$\sum_{k \in C} \sum_{i \in S \cup R \cup \{k\}} x_{i,j}^{(k)} = 1, \forall j \in S \cup R \tag{8}$$

$$\sum_{i \in S \cup R \cup \{k\}} x_{i,\omega}^{(k)} - \sum_{j \in S \cup R} x_{\omega,j}^{(k)} \leq 1, \forall \omega \in S \cup R, \forall k \in C \tag{9}$$

$$\sum_{i \in S \cup R/Z} \sum_{j \in Z} x_{i,j}^{(k)} > 0, Z = \{z \in Z : \sum_{j \in S \cup R} x_{j,z}^{(k)} > 0\}, \forall k \in C \tag{10}$$

Ограничение (7) запрещает, чтобы у узла графа G было более чем одна выходящая дуга. Ограничение (8) запрещает узлу иметь более чем одну входную дугу. Ограничение (9) указывает на то, что у узла количество входных дуг не может быть меньше количества выходных дуг (это ограничение учитывает тот факт, что транспортное средство может покинуть пункт назначения, если только он посетил этот узел). Ограничение (10) исключает локальные циклы.

Следующие ограничения синхронизируют значения переменных X и $\vec{}$для каждого запроса Q Q и запрещают посещение получателя перед отправителем:

$$x_{s_q}^{(k)} + x_{r_q}^{(k)} = 2 \cdot y_q^{(k)}(X), \forall k \in C, q \in Q \tag{11}$$

$$y_q^{(k)}(X) \cdot (\tilde{t}_{r_q}^k(X^{(k)}) - \tilde{t}_{s_q}^k(X^{(k)}) \geq 0, \forall k \in C, q \in Q \tag{12}$$

Определим ограничения для учета грузоподъемности транспортных средств и временные окна:

$$\sum_{j \in Q} \omega_j \cdot y_j^k \leq L_k, \forall k \in C \tag{13}$$

$$t_s^q \leq \tilde{t}_{s_q}^k(X^{(k)} \leq t_s^q + \Delta t_s^q, \forall q \in Q, \tag{14}$$

$$t_r^q \leq \tilde{t}_{r_q}^k(X^{(k)} \leq t_r^q + \Delta t_r^q, \forall q \in Q, \tag{15}$$

Описание алгоритма

Представим множество матриц $\{X^k\}$ в виде вектора, определенного на гиперкубе $E^\eta = \{0.1\}^\eta$, где $\eta = n \cdot (2m+1) \cdot 2m$. Цель задачи состоит в минимизации следующей целевой функции:

$$F(\vec{u}) \to min, \vec{u} \in E^\eta \tag{16}$$

Обозначим через $\delta(\vec{u}, \vec{v})$ расстояние Хэмминга между $\vec{u}$ и $\vec{v}$. Через $N_l(\vec{u})$ обозначим окрестность точки $\vec{u}$ радиусом l [4]:

$$N_l(\vec{u}) = \{\vec{v} \in E^\eta : \delta(\vec{u}, \vec{v}) \leq l\}, l = 1, \bar{}\eta \tag{17}$$

При $l = \eta$ множество $N_l(\vec{u})$ для любого вектора $\vec{u}$ совпадает с множеством E^η и поиск в этой окрестности вектора с минимальным значение целевой функции эквивалентно решению исходной задачи. Стандартный алгоритм локального поиска начинается с выбранного случайным образом вектора $\vec{u^0}$. На i шаге алгоритма выполняется переход от текущего вектора в минимум окрестности:

Когда $l = \eta$, тогда множество $N_l(\vec{u})$ для любого вектора $\vec{u}$ совпадает с множеством E^η и поиск в этой окрестности минимума эквивалентно решению исходной задачи. Классический алгоритм локального поиска начинает поиск оптимума с инициализации вектора $\vec{u^0}$. На i шаге алгоритма алгоритма выполняется перемещение текущего вектора в минимум его окрестности:

$$F(\vec{u}^{i+1}) = min\{F(\vec{v}) : \vec{v} \in N_l(\vec{u}^i)\} \tag{18}$$

Алгоритм завершается в локальном оптимуме, когда $F(x^{\vec{i+1}}) = F(\vec{x^i})$. В задачах маршрутизации, как правило, имеется большое множество локальных оптимумов и только один из них носит глобальный характер:

$$F_{opt} = min\{F(\vec{v}) : v \in E^{\eta}\} \tag{19}$$

Для того, чтобы убедиться, что алгоритм не остановится в локальном минимуме, и перемещается от одного локального минимума в другой, с окрестности удаляется центральная точка и при поиске минимума окрестности применяется следующее правило. Пусть $l = 2$ и при переходе от $u^{\vec{i+1}}$ к $\vec{u^i}$ изменяются значения в координате $(u^i_\lambda, u^i_\omega)$. Алгоритм хранит такие пары за последние h шагов и на следующем этапе запрещает движение в этих направлениях. Упорядоченный список таких пар:

$$\phi^i = \{(u^i_\lambda, u^i_\omega), (u^{i-1}_\lambda, u^{i-1}_\omega), \cdots, (u^{i-h+1}_\lambda, u^{i-h+1}_\omega)\} \tag{20}$$

является списком запретом. При создании списка всех пар и пару $(u_\lambda, u_\omega), \lambda \neq \omega$ не запрещает движение пар (u_λ, u_λ) and (u_ω, u_ω). Когда $l > 2$ аналогичным образом построены три координаты, четыре и т.д. Набор неограниченная векторов обозначим через $N_l(\vec{u^i}, \vec{\phi^i})$. Для того чтобы поиск был эффективным, то целесообразно использовать малые значения h и контролировать этот параметр в ходе алгоритма.

```
1  function TabuSearch(u^0, l, p, h)
2  // Initialize variables:
3  u^opt = u^0  F^opt = F^0  φ^0 = ∅  i = 0
4  while the breakpoint is not triggered do
5  |  N_l = N_l(u^i, φ^i, p, h)
6  |  if N_l ≠ ∅ then
7  |  |  u^{i+1} = u^i
8  |  |  i = i + 1
9  |  |  goto 5
10 |  else
11 |  |  // find optimum into the neighborhood N_l:
12 |  |  u^{i+1} : F(u^{i+1}) = min{F(y) : y ∈ N_l}
13 |  end
14 |  if F(u^{i+1}) < F_opt) then
15 |  |  F_opt = F(u^{i+1})
16 |  |  u^opt = u^{i+1}
17 |  end
18 |  φ^{i+1} = update(φ^i)
19 |  i = i + 1
20 end
21 return u^opt
```

Algorithm 1: Pseudo-code for probabilistic Tabu Search algorithm.

Обозначим через $N_l(\vec{u^i}, \vec{\phi^i}, p)$ вероятностный край, который идет от детерминированной $N_l(\vec{u^i}, \vec{\phi^i})$ следующим образом. Каждый вектор $\vec{v} \in N_l(\vec{u^i}, \vec{\phi^i})$ с вероятностью p включенных в окрестности $N_l(\vec{u^i}, \vec{\phi^i}, p)$ независимо от других точек. Обратите внимание, что этот набор может быть пустым или содержать только одну точку. Общая схема вероятностного алгоритма поиска со списком запретов упоминается как псевдо-кода в алгоритме
представлена на схеме 1.

По мере того как критерий останова основан на общем количестве шагов N_stop, в ходе которого не изменяет значение F_{opt}. Значения l, p, h являются настроечными параметрами алгоритма. Их выбор зависит от размерности задачи.

В представленной схеме, предполагалось, что величина h (размерность списка запретов) не изменяется в ходе алгоритма. Это создает определенные трудности в реализации схемы, поскольку неизвестно, каким должен быть размер списка запретов. При малых h алгоритм может зациклиться. При больших h поиск становится неэффективным.

Инициализация

Для того, чтобы использовать алгоритм поиска с запретами, необходимо построить первоначальное решение. Для получения первоначального решения $\vec{u^0}$ можно воспользоваться эвристическим методом конструирования маршрута. Суть этих методов состоит в следующем. Для каждого автомобиля последовательно строится маршрут путем добавления еще не рассмотренных клиентских заявок по заданным правилам. В нашем случае, для сохранения порядка посещения вершин, под заказом понимается пара "отправитель-получатель".

Самый известный метод конструирования маршрута был предложен Соломоном в 1987 году [**?**, **?**] и в литературе встречается как метод Соломона.Данный алгоритм является достаточно быстрым (вычислительная сложность $O(n^3)$ [**?**]), поэтому его удобно использовать для инициализации вектора $\vec{u^0}$.

В начале работы алгоритма происходит добавление по одному запросу на маршрут в соответствии с временными ограничениями. Выбор первой заявки происходит случайным образом, или выбирается та, которую необходимо выполнить раньше других. Далее каждая возможная заявка u из множества нерассмотренных заявок Q (множество $Q' \in Q$) оценивается на предмет вставить ее в начало маршрута, конец или между двумя соседними узлами маршрута. Для выбора места вставки используется следующий критерий:

$$c(k, v_i, u, v_{i+1}) = \min = \begin{cases} \min_{q=2,\ldots,n^k}(\Omega^k_{q-1,u} + \Omega^k_u + \Omega^k_{u,q} - \mu\Omega^k_{q-1,q}), \\ \Omega^k_{k,u} + \Omega^k_{u,q} + \Omega^k_u\mu\Omega^k_{q,q}), q = 1 \\ \Omega^k_{q,u} + \Omega^k_u, q = n^k \end{cases} \quad (21)$$

где:
k – маршрут транспортного средства $k \in C$;
n^k - количество узлов в маршруте;

μ –настроечный параметр, $\mu \geq 0$;
$\Omega_u^k = \Omega^k[\vec{P}_{s_q}, \vec{P}_{r_u}]$ – вариант вставки новой заявки между уже существующими отправителем и получателем;
$\Omega_{u,q}^k = \Omega^k[\vec{P}_{r_u}, \vec{P}_q]$ – вариант вставки новой заявки на начало маршрута;
$\Omega_{q,u}^k = \Omega^k[\vec{P}_q, \vec{P}_{s_u}]$ – вариант вставки новой заявки в конец маршрута.

Добавляется та заявка, у которой $c(k, v_i, u, v_{i+1})$ будет минимальным:

$$u^* : c(k, v_i, u^*, v_{i+1}) = \min_u [c(k, v_i, u, v_{i+1})] \tag{22}$$

В результате u^* добавляется к текущему маршруту k на наиболее выгодную позицию.

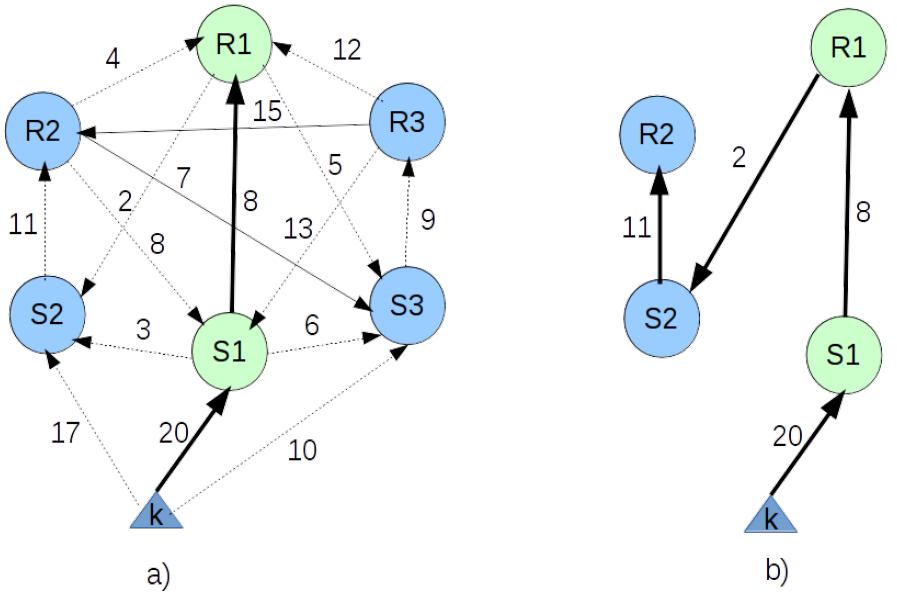


Рис. 2. Пример выполнения одного шага методом Соломона

На рис. 2 представлен взвешенный граф, на примере которого показано выполнение одного шага методом Соломона:
a) исходный граф, на котором показаны варианты добавления новых заявок на маршрут;
b) маршрут, который был построен методом Соломона после выполнения одного шага.

Пример. Изначально данна заявка (которая состоит из отправителя и получателя). Значение функции стоимости Ω представлены на дугах графа (рис.2). Необходимо добавить новые заявки в маршрут, используя критерии 21 і 22. Коэффициент $\mu = 0.5$.

*Решение.*Рассмотрим процесс добавления новых заявок. Из двух предложенных заявок ($\{S_2, R_2\}, \{S_3, R_3\}$) выберем ту, которую нужно добавить в первую очередь. В зависимости от места куда можно добавить заявку (начало, конец, середина) выберем нужный критерий. Таким образом, найдем оптимальное решение вставки новой заявки $\{S_2, R_2\}$:
$\min[c(k, S_1, \{S_2, R_2\}, R_1), c(k, \{S_2, R_2\}, S_1, R_1), c(k, S_1, R_1, \{S_2, R_2\})] = \min[3 + 11 + 4 - 0.5 \cdot 8; 17 + 11 + 8 - 0.5 \cdot 20; 2 + 11] = \min[14; 26; 13]$
Найдем оптимальное решение вставки новой заявки $\{S_3, R_3\}$:
$\min[c(k, S_1, \{S_3, R_3\}, R_1), c(k, \{S_3, R_3\}, S_1, R_1), c(k, S_1, R_1, \{S_3, R_3\})] = \min[6 + 9 + 12 - 0.5 \cdot 8; 10 + 9 + 13 - 0.5 \cdot 20; 5 + 9] = \min[23; 22; 14]$.
Из найденных решений выберем решение с наименьшим значением $c(k, v_i, u, v_{i+1})$. В нашем примере это $c(k, S_1, R_1, \{S_2, R_2\} = 13$.

Таким образом, новый маршрут будет состоять из начальных заявок и новой заявки $\{S_2, R_2\}$,которая будет добавлена в конец маршрута.

Ответ: в маршрут $\{S_1, R_1\}$ необходимо подать заявку $\{S_2, R_2\}$, которая будет размещена в конце.

На рис. 3 представлены варианты добавления в маршрут новой заявки $\{S_2, R_2\}$:
a) вариант добавления заявки $\{S_2, R_2\}$ в конец маршрута;
b) вариант добавления заявки $\{S_2, R_2\}$ в начало маршрута;
c) вариант добавления заявки $\{S_2, R_2\}$ между узлами $\{S_1, R_1\}$.

На рис. 4 представлены варианты добавления в маршрут новой заявки $\{S_3, R_3\}$:
a) вариант добавления заявки $\{S_3, R_3\}$ в кінець маршруту;
b) вариант добавления заявки $\{S_3, R_3\}$ в початок маршруту;
c) вариант добавления заявки $\{S_3, R_3\}$ между узлами $\{S_1, R_1\}$.

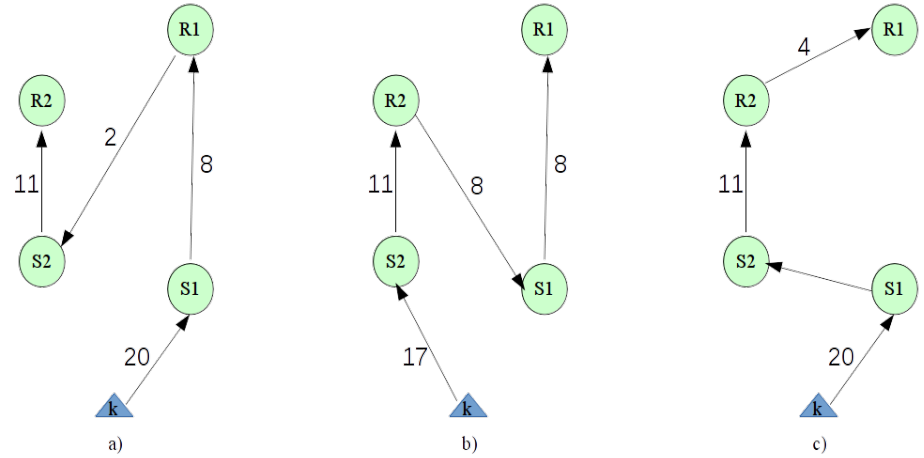


Рис. 3. Варианты добавления в маршрут заявки $\{S_2, R_2\}$

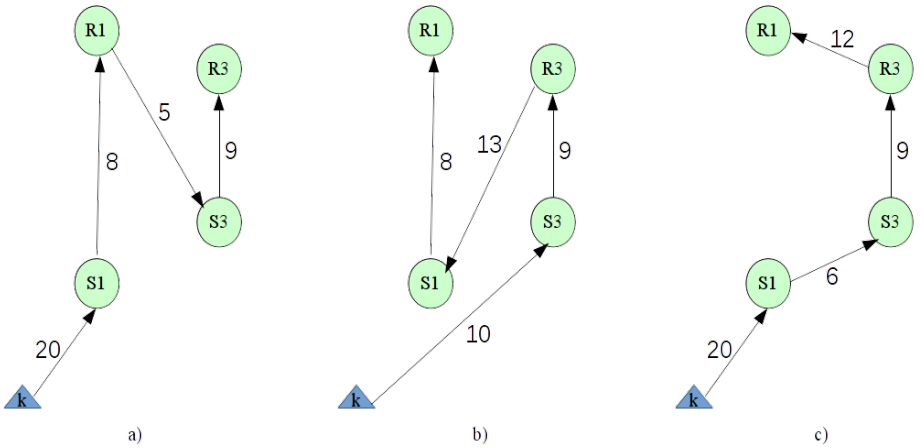


Рис. 4. Варианты добавления в маршрут заявки $\{S_3, R_3\}$

Метод формирования окрестности

Во время работы алгоритма имитации отжига необходимо проводить просмотр окрестности $N_l(\vec{u^i})$ текущей точки $\vec{u^i}$. В ходе вычисления маршрута в окрестность попадает достаточно большое количество точек. Но в связи с особенностями ограничений нашей модели многие из этих точек не является решением задачи. Поэтому в связи с особенностями рассматриваемой дискретной задачи мы использовали следующий прием определения окрестности.

Пусть $x^k(v)$ – индекс узла v, $v \in R \cup S$ в маршруте автотранспортного средства $k \in C$. Каждая заявка состоит из пары $\{S, R\}$, где S – отправитель, а R – получатель. В процессе формирования окрестности, должна сохраняться целостность маршрута, то есть выполняться ограничения (7 –10). Над заявками маршрута $Route_i$, $i = 1 \dots n$выполняются операции перестановки. Полученные в ходе перестановок векторы $\vec{u}$ и будет окрестностью $N_l(\vec{u^i})$. Для перестановки использовались операции перемещения и поглощения.

Операция поглащения представлена на рис.5(a), на котором представлены два маршрута $Route_1$ і $Route_2$. Маршрут $Route_1$ состоит из улов $\{C_1, S_1, R_1, S_2, R_2, S_3, R_3\}$, а маршрут $Route_2$ состоит из узлов $\{C_2, S_4, R_4, S_5, R_5, S_6, R_6\}$. Во время выполнения операции поглощения $\{S_1, R_1\}$ в маршруте $Route_1$, вставка данной заявки в маршрут $Route_2$ може бути виполнена на $n + 1$ позиций.

Операция перемещения представлена на рис.5(b). На данном рисунке представлен маршрут $Route_1$, который состоит из узлов $\{C_1, S_1, R_1, S_2, R_2, S_3, R_3\}$. Для сохранения целостности маршрута: узел S_i (отправитель) может быть посещен только до R_i (получатель),исходя из этого во время операции перемещения, узел S_i может перемещаться пока будет выполняться условие $0 < x^{new}(S_i) < \widetilde{u}(R_i)$,а узел R_i может перемещаться пока будет выполняться условие $n^k \geq x^{new}(R_i) > \widetilde{u}(S_i)$.

Анализ параметров алгоритма

The Tabu Search алгоритм имеет два основных настроечных параметра: h – размер списка запретов и $N_{neighbors}$ –Предел допускаемого подсчет размера окрестности $N_l(\vec{u})$. Для экспериментов, мы построили тестовые задания на основе проблем маршрутизации мощности, разработанных Breedam, Fisher, Christofides and Eilon. Для определения тенденций во времени и стоимости решения мы используем экспоненциальное сглаживание с коэффициентом 0.1.

В этом эксперименте мы используем две различные точки останова стратегии:

1. *StepCount* Стратегия завершения - заканчивается, когда количество шагов было достигнуто;
2. *TimeSpent* Стратегия завершения - заканчивается, когда количество времени было достигнуто.

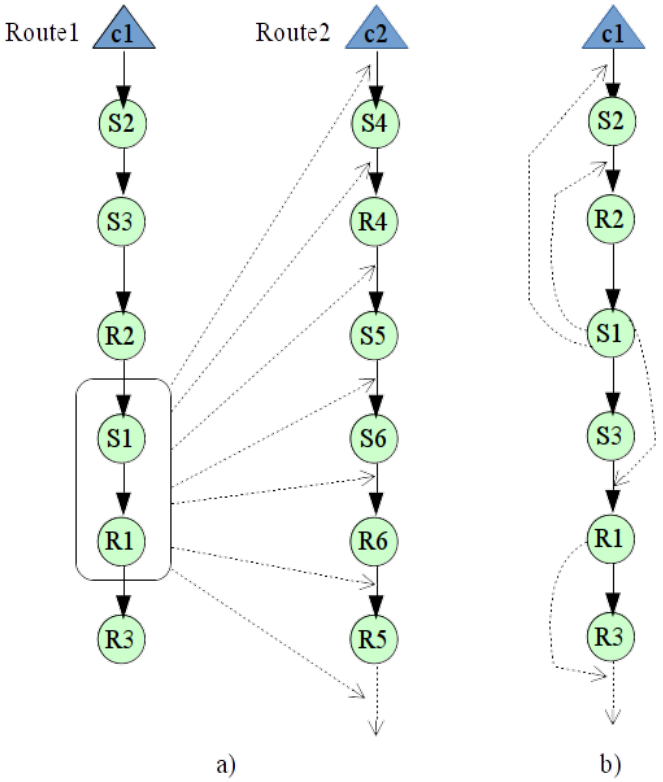


Рис. 5. Пример операции a)поглащения и b)перемещения

Как и следовало ожидать, при больших значениях параметра застревание происходит в локальный оптимум из-за большого количества ограничений движения в космосе. В противном случае, если вы выбрали слишком маленький размер Табу алгоритм может все еще застревают в локальный оптимум. В первом вычислительного эксперимента мы сделали находить оптимальные решения для различных значений параметра размера Табу из интервала $h \in [0, 40]$ (рис. 6 рис. 7).

Во втором эксперименте (смотри рисунок 8) мы построили зависимость между параметром $N_{neigh.}$ просматриваемой размера решения окрестности $N_l(\vec{u})$ используя $TimeSpent$ стратегии завершения. В этом эксперименте мы сделали находить оптимальные решения для различных значений параметра размера окрестности от интервала $N_{neigh.} \in [750, 850]$.

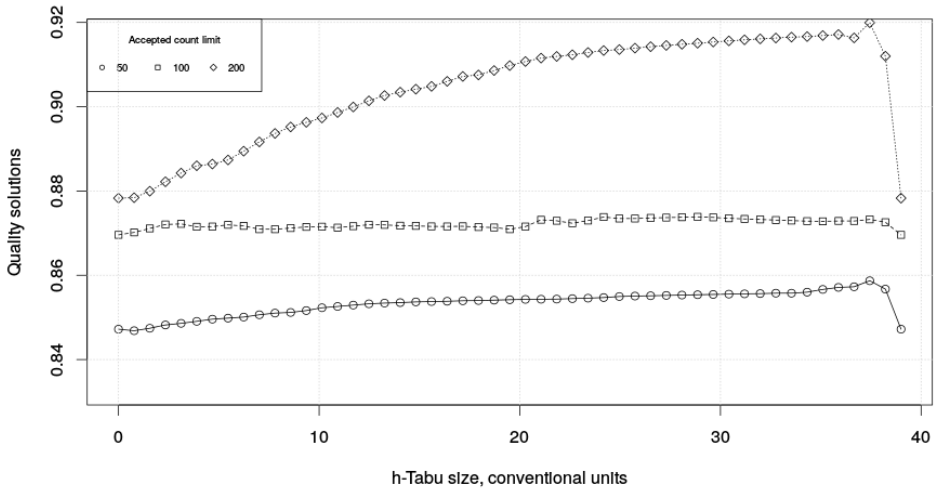


Рис. 6. The dependence of the quality of solutions on the size of the tabu list when using *StepCount* termination strategy

Modification

Первая обобщенная схема выглядит следующим образом. Во-первых, множество решений строится Соломона. Тогда каждое решение улучшается поиск с запретами и выбрать оптимальное решение в соответствии с целевой функцией.

Модификация схемы основана на гипотезе «О большой долине"[9]. Согласно этой гипотезе, в среднем локальных оптимумов расположены гораздо ближе к глобальным, чем случайно выбранной точке. Существует определенная концентрация локального оптимума в небольшой части допустимой области, которая образно называется большая долина. Если это предположение верно, то желательно, чтобы вспомнить лучшие решения и на их основе разработать новое оригинальное решение. Мы используем эту идею для решения маршрутизации Courier проблемы доставки.

Перейдем к описанию алгоритма в схеме 2. Пусть U^{opt} является отсортированный массив оптимальных решений по значению целевой функции по возрастанию, то есть:

$$F(u_1^{opt}) \leq F(u_2^{opt}) \leq \ldots < F(u_i^{opt}) \leq \ldots \leq F(u_{sizeof(U^{opt})}^{opt}) \tag{23}$$

Каждое решение u_i^{opt} попадает в популяции с заданной вероятностью p_u^i, вероятность выбора уменьшается с увеличением порядкового номера:

$$p_u^1 = 1 > p_u^2 > \ldots > p_u^i > \ldots > p_u^{sizeof(U^{opt})}, p_u^{sizeof(U^{opt})} > 0 \tag{24}$$

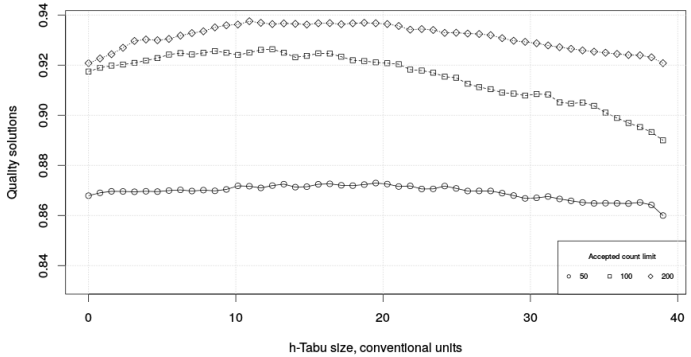


Рис. 7. Зависимость качества решений от размера списка при использовании табу *TimeSpent* стратегии завершения

```
1  function constructSolution( U^opt, p_u, n_s^min)
2  // U^opt – a sorted set array of optimal solutions
3  if sizeof(U^opt) < n_s^min then
4  |  // construct solution using heuristics(for example, Solomon alg.)
5  |  u^0 = heuristicsConscruction()
6  |  return u^0
7  end
8  while i < sizeof(U^opt) do
9  |  if random(0.0, 1.0) > p_u^i then
10 |  |  i = i + 1
11 |  |  goto 8
12 |  end
13 |  R_i =randomly select a route from u_i^opt solution; u^0 = u^0 ∪ R_i i = i + 1
14 end
15 if u^0 isnotcomplete then
16 |  u^0 = heuristicsConscruction(u^0)
17 end
18 return u^0
```

Algorithm 2: Pseudo-code for heuristics construction algorithm.

Затем она пересекает решения по следующему правилу: с первым решением случайным образом выбирается маршрут R_1. Тогда другое решение выбирается из маршрута R_2 который не включает в себя клиентские запросы от первого маршрута и т.д. Если клиентские запросы не были включены в решение u^0, t то эти запросы добавляются в алгоритм построения эвристики (например, Соломон алгоритм).

Теперь мы опишем основную функцию $SolveCDP()$ модифицированного

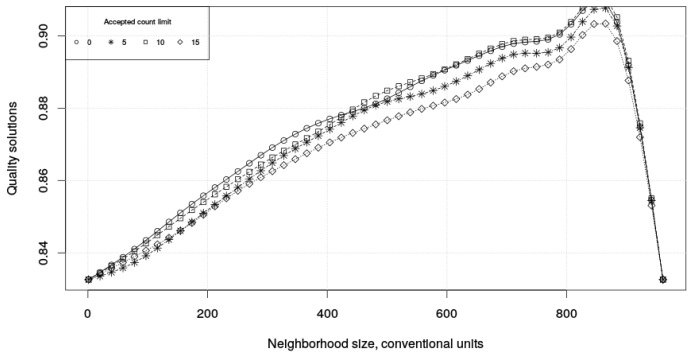


Рис. 8. Зависимость качества решений от размера окрестности при использовании *TimeSpent* стратегии завершения

алгоритма представлено в виде псевдокода на схеме 3. В этой функции с помощью цикла является последовательное формирование массива из лучших решений U^{opt} с помощью алгоритма TabuSearch.

На основании результатов экспериментов, описанных в предыдущем параграфе, мы решили установить параметры алгоритма не являются фиксированными значениями, но, как случайной величины заданного периода. Например, размер списка h определяется как интервал $[10, 35]$(see pict. 6 and pict. 7) и размер окрестности $l = N_{neighbordhood}$ определяется как интервал $[600 \cdot n, 850 \cdot n]$, где n – размерность задачи (см. рисунок 8).

Пусть $\vec{\psi}_i = (\psi_i^1, \psi_i^2, \psi_i^3) = (\tilde{l_i}, \tilde{p_i}, \tilde{l_h})$ – набор значений параметров конфигурации TabuSearch, который использовался для решения задачи на этапе i.

Пусть $[\psi_{begin}^j, \psi_{end}^j]$ – оптимальный интервал TabuSearch j -parameter. (For example: if $j = 3$, then: $[\psi_{begin}^3, \psi_{end}^3] = [h_{begin}, h_{end}] = [10, 40]$). Эти интервалы являются исходные данные, и они установлены в блоке инициализации.

Настроечные параметры ψ_i^j выбираются случайным образом из интервала $[\psi_{begin}^j, \psi_{end}^j]$ в соответствии с законом распределения. Мы предложили построить плотность распределения $f_j(\psi_i^j)$ случайной величины ψ_i^jследующим образом. Во-первых, мы определяем опорные точки:

$$\rho_i^j = \frac{F(u_1^{opt})}{avg[F(u_k^{opt})]}, \forall k : \psi_k^j = \psi_i^j \tag{25}$$

Число опорных точек n_ρ меньше, чем множество оптимальных решений, так как набор опорных точек, удаляются те же точки ($n_\rho \leq n_u$).

Основная идея заключается в том, что плотность распределения должна быть больше в тех точках (параметры ψ_i^j) в которых высококачественные решения, чем точек, в которых низкое качество решений. Таким образом, мы

представили функцию распределения $f_j(z)$ в виде суммы ядерных функций:

$$f_j(z) = \alpha_j * \sum_{i=1}^{n_\rho} \rho_i^j * K(z - \psi_i^j) \tag{26}$$

где: $K(x)$ – ядерная функция, α – нормирующий параметр

Мы выбрали в качестве ядра функции параболического типа (известный как функция Епанечникова), так как в ходе экспериментов лучшие результаты были получены с помощью этой функции:

$$K(z) = 3/4 \cdot (1 - z^2) \tag{27}$$

function SolveCDP(n_s^{min})
// Initialize variables:
$\psi = \emptyset$, $p_u = \emptyset$, $U^{opt} = \emptyset$, $i = 0$
while *the breakpoint is not triggered* **do**
  // set TabuSearch parameters
  for $j = 1 \dots 3$ **do**
    // f_j – the density distribution of the random
    // variable ψ^j (TabuSearch parameter l, p or h)
    $\psi_i^j = random(\psi_{begin}^j, \psi_{end}^j, f_j)$
  end
  // Make solution using euristics construction algorithms
  $u^0 = constructSolution(U^{opt}, n_s^{min})$
  // Improve solution u^0 using TabuSearch algorithm
  $u_i^{opt} = TabuSearch(u^0, \vec{\psi_i})$
  $U^{opt} = U^{opt} \cup \{u_i^{opt}\}$
  // Sort ascending optimums U of the objective function
  $U^{opt} = sort(U^{opt})$
  $p_u^{i+1} = update(p_u^i)$
  i = i + 1
end
return u_1^{opt}

Algorithm 3: Pseudo-code for modified tabu-search algorithm.

Параметр α_j была введена в функции плотности для выполнения условия нормировки:

$$\int_{\psi_{begin}^j}^{\psi_{end}^j} f_j(z)dz = 1 => (\alpha_j)^{-1} = \sum_{i=1}^{n_\rho} \rho_i^j \cdot \int_{\psi_{begin}^j}^{\psi_{end}^j} K(z - \psi_i^j)dz \tag{28}$$

Параметр α_jможно записать в явном виде:

$$(\alpha_j)^{-1} = \frac{3}{4} \cdot \sum_{i=1}^{n_\rho} \rho_i^j \cdot [(1 - (\psi_i^j)^2) \cdot (\psi_{end}^j - \psi_{begin}^j) + \\ + \psi_i^j \cdot ((\psi_{end}^j)^2 - (\psi_{begin}^j)^2) - 1/3 \cdot (\psi_{end}^j)^3 - (\psi_{begin}^j)^3)] \tag{29}$$

На рисунке 9 результаты эксперимента, в котором были представлены зависимость качества получаемых решений с помощью классических и модифицированных алгоритмов и времени переломной точкой. Как видно из графика, с увеличением времени работы расчетной схемы, качество получаемых решений с использованием модифицированного алгоритма растет быстрее, чем при использовании классического алгоритма Табу поиска.

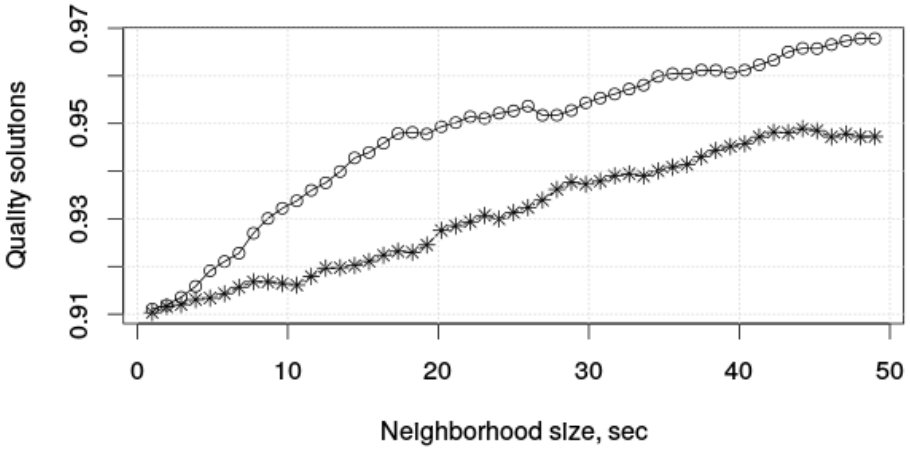


Рис. 9. Зависимость качества растворов и времени Breaking Point с помощью классических и модифицированных алгоритмов с помощью *TimeSpent* стратегии завершения

Conclusion

В ходе исследования был исследован и реализован алгоритм поиска с запретами решить проблему доставки статической маршрутизации Courier с нехваткой времени. Кроме того, модифицированный алгоритм был разработан на основе алгоритма Табу поиска, чтобы улучшить качество решений. Модифицированный алгоритм дал лучшие решения с точки зрения баланса между количеством транспортных средств и стоимости пройденного. На практике

этот алгоритм может быть использован для поддержки принятия решений в интеллектуальных системах для повышения качества обслуживания клиентов и сокращения ждут время, чтобы использовать транспортное средство, это позволит сократить расходы на топливо и амортизацию транспорта.

Анализ параметров реализованных алгоритмов позволили определить их оптимальные значения для этого класса задач маршрутизации. С помощью модифицированного алгоритма были найдены решения модельных задач, которые в большинстве случаев имеют приемлемое отклонение от глобального оптимума.

Список литературы

[1] F. Ordonez, Chen Wang, A New Approach for Routing Courier Delivery Services //METRANS Transportation Center:University of Southern California Los Angeles, 2012, 81–115.

[2] R. Masson, F. Lehued, O. Peton,The dial-a-ride problem with transfers. // Computers and Operations Research, Vol. 41, 2014, p. 12–23.

[3] T. Babb, Pickup and Delivery Problem with Time Windows // Coordinated Transportation Systems: The State of the Art. Department of Computer Science University of Central Florida Orlando, Florida, 2005, 38 p.

[4] R. Shafeyev ,L. Lyubchik, A some realization of Tabu Search algorithm for Solving the Transportation Problem with Time Constraints // Vestnik NTU "KhPI". – Kharkov: NTU "KhPI 2013. – No3 (977). – p. 35–39.

[5] R. Shafeyev. Java-based optimixation framework for solving routing problems. url: http://jlogistics.net, 2015.

[6] W Barnes, Solving the pickup and delivery problem with time windows using reactive tabu search // Transportation Research Part B: Methodological, Vol. 34 Issue 2, 2000, p. 107–121.

[7] B. Coltin, M. Veloso, Scheduling for Transfers in Pickup and Delivery Problems with Very Large Neighborhood Search // The Twenty-Eighth Conference on Artificial Intelligence, Quebec City, Canada, 2014, 7 p.

[8] E. Goncharov, Y Kochetov, Probabilistic search with exclusions for discrete unconstrained optimization //Discrete Analysis and Operations Research, Moscow, Serial 2. Vol. 9, 2002, p. 13-30.

[9] Y Kochetov, Probabilistic methods of local search for discrete optimization problems //Discrete Mathematics and Its Applications: Proceedings of youth lectures and scientific schools in discrete mathematics and its applications, Publishing House of the Center for Applied Research at the Mechanics and Mathematics. Faculty of Moscow State University, 2001, p. 84-117.

[10] O. Braysy, M. Gendreau, Vehicle Routing Problem with Time Windows, Part I: Route Constuction and local algorithms // Transportation science Vol.39 No. 1, 2005, p. 104-118.

[11] V. Goldberg, R. Kennedy, An Efficient cost scaling algotirhm for the assignment problem, Math. Program., 1995, p. 153–177.

[12] N. Christofides, S. Eilon, An algorithm for the vehicle dispatching problem //Operational Research Quarterly, 20, 1969, p. 309–318.
[13] B. Golden, E. Wasil, J. Kelly, I-M. Chao. The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and computational results. In T. Crainic and G. Laporte, editors // Fleet Management and Logistics, Kluwer, Boston, 1998 p. 33–56.
[14] E. Taillard. VRP benchmarks.
url: http://mistic.heig-vd.ch/taillard/problemes.dir/vrp.dir/vrp.html, 1993.
[15] A. Attanasio, J. Bregman,G. Ghiani, E. Manni. Real-time fleet management at Ecourier Ltd. Dynamic Fleet Management, volume 38 of Operations Research/Computer Science Interfaces, chapter 10, p.219–238, 2007.
[16] A. Beaudry, G. Laporte, T. Melo, Nickel. Dynamic transportation of patients in hospitals. Berichte des Fraunhofer ITWM, Nr. 104 :p.1–34, 2010.
[17] M.Romero, L.Sheremetov and A.Soriano. A genetic algorithm for the pickup and delivery problem: An application to the helicopter offshore transportation. In Theoretical Advances and Applications of Fuzzy Logic and Soft Computing, volume 42 of Advances in Soft Computing, 2007, p. 35–44.
[18] M.Romero, L.Sheremetov and A.Soriano. Yannick Kergosien, Christophe Lent, D. Piton, Jean-Charles Billaut. A tabu search heuristic for a dynamic transportation problem of patients between care units. 29 pages. 2010.
[19] H. Sarak, A. Satman. The degree-day method to estimate the residential heating natural gas consumption in Turkey: a case study. Pergamon, Energy 28 (2003) 929–939.