Санкт–Петербургский государственный университет
Кафедра Математического Моделирования Энергетических Систем

Поляков Иван Михайлович

Выпускная квалификационная работа

Эвристические алгоритмы в логистических системах

Направление 01.03.02
«Прикладная математика, фундаментальная информатика и программирование»

Научный руководитель:
доцент, кандидат физ.-мат. наук
Лежнина Елена Александровна

Рецензент:
кандидат физ.-мат. наук,
старший инженер-программист ООО "Интермедиа"
Перцовский Александр Константинович

Санкт-Петербург
2021 г.

Содержание

Введение

Ежегодно спрос на транспортно-логистические услуги неимоверно повышается во всем мире. Связано это, в первую очередь, с ростом потребительского спроса на разного рода товары. Также не стоит игнорировать факт повышения загруженности автомобильных дорог, что может приводить к запаздыванию в установленные временные интервалы доставки, и, как следствие, к убыткам. В связи с этим, проблема решения задачи транспортной маршрутизации (*Vehicle Routing Problem - VRP*) становится более востребованной.

О важности рассмотрения этой задачи в повседневной жизни рассказывается в статье [1], где впервые была поставлена математическая формулировка, а также решена задача о поставке бензина на заправочные станции. Классической формулировкой задачи транспортной маршрутизации является следующее определение: транспортным средствам необходимо доставить товары потребителям, минимизируя затраты на перевозку этих самых товаров. Решением данной задачи будет является множество маршрутов, где:

- маршрут начинается и заканчивается в одном и том же месте - депо;
- все потребители должны быть обслужены и посещены единожды.

При постоянных местоположениях клиентов и депо переменной, в данном случае, является лишь затрата на перевозку. Эти затраты могут быть уменьшены как с помощью оптимизации пройденного расстояния, так и с помощью уменьшения количества транспортных средств [2].

С момента публикации статьи [1] была проведена большая исследовательская работа: выведены подзадачи, проведена их классификация [3], установлены эффективные методы и алгоритмы, позволяющие получить оптимальное или "почти оптимальное"решение, доказана *NP*-трудность [2], [4].

В настоящее время подавляющее число ограничений формулируют разные задачи маршрутизации транспорта, отличные от классической. Таким образом, наиболее востребованными задачами являются:

1. С ограниченной грузоподъемностью транспортного средства – *Capacitated VRP*;

2. С ограничением на период обслуживания – *VRP with Time Windows*;

3. С возможностью возврата товара – *VRP with PickUps and Deliveries*;

4. С возможностью возврата товара после обслуживания всех клиентов – *VRP with Backhauls*;

5. С возможностью обслуживать клиентов разными видами транспортных средств – *Split Delivery VRP*;

6. С возможностью довоза товара за определенный период – *Periodic VRP*;

7. Со случайными данными – *Stochastic VRP*;

8. С возможностью использования нескольких депо – *Multiple Depot VRP*;

9. С возможностью добавочной загрузки транспортного средства – *VRP with Satellite Facilities*.

В данной работе исследуется эффективность эвристических методов и алгоритмов задачи CVRPTW, которая является объединением двух задач: CVRT и VRPTW.

Таким образом, задача формулируется следующим образом:
Транспортные средства, имеющие ограниченную грузоподъёмность, должны доставить товары потребителям в определенные временные промежутки, минимизируя затраты на перевозку.

Постановка задачи

Целью данной работы является исследование эффективного решения задачи транспортной маршрутизации с временными окнами и ограниченной грузоподъемностью транспорта на сетях больших и малых размерностей на известных примерах. Для достижения цели были поставлены следующие задачи:

1. Исследование решений оптимизационных задач, генерируемых эвристическими алгоритмами;
2. Освоение принципов работы эвристических методов решения транспортных задач маршрутизации;
3. Анализ результатов.

Обзор литературы

Повышение эффективности транспортировки для распределительных центров во многом связано с проблемой маршрутизации транспортных средств (VRP). VRP - одна из самых известных задач комбинаторной оптимизации. Dantzig и Ramser в статье [1] представили VRP на основе реальной проблемы поставки бензина на заправочные станции. Тогда были предложены формулировка математического программирования и алгоритм для VRP. С тех пор задача породила хорошо известную и важную область операционных исследований.

В этой области существуют множество проблем с конкретными ограничениями. Например, в гетерогенной VRP [5] задачи по доставке были назначены более чем одному типу транспортных средств; Solomon в [6] представил проблему с временными окнами ($VRPTW$), в которой клиенты должны обслуживаться в определенное время; в $PVRP$ план доставки включает в себя несколько дней за раз [7]. В данной работе, фокус будет смещен на $CVRPTW$.

В 1981 году Lenstra и Kan в статье [4] изучили сложность VRP и пришли к выводу, что все задачи подобного типа являются NP-сложными. В 1988 году Solomon и Desrosiers в [8] также доказали, что $CVRPTW$ представляет собой NP-сложную задачу. В результате только небольшое число случаев $CVRPTW$ может быть решено и доказано как оптимальное. Ниже приведена краткая история развития VRP.

Десятилетие	Событие
1950	Dantzig и Ramser (1959) представили проблему маршрутизации транспорта; Решено несколько примеров малых размерностей (от 10 до 20).

1960	Clarke и Wright (1964) предложили алгоритм построения маршрутов [9]; 2-opt и 3opt были применены для улучшения решения (1969) [10]; Решено несколько примеров малых размерностей (от 30 до 100).
1970	Разработаны двухэтапные эвристики (Gillett и Miller, 1974) [11]; Для некоторых примеров малых размерностей (20-25) было найдено оптимальное решение.
1980	Разработана интерактивная эвристика (Cullen и Jarvis, 1981) [12]; Для некоторых примеров с 50 клиентами были найдены оптимальные решения.
1990	Была предложена мета-эвристика для VRP; Для нескольких примеров до 100 клиентов были найдены оптимальные решения.

Таблица 1: Краткая история VRP [29]

Проблема маршрутизации транспортного средства с ограниченной грузоподъёмностью и временными окнами - это $CVRP$ с дополнительными ограничениями по времени. В реальном мире некоторые клиенты могут обслуживаться только в определенный период времени, называемый временными окнами. Если транспортные средства прибудут к клиенту раньше, они могут дождаться нижней границы временного окна, так как ожидание не требует дополнительных затрат. Если же транспорт прибывает к клиенту после временного окна, то, поскольку клиент недоступен, транспортное средство не может выполнить свою работу, и следует учитывать дополнительные штрафные расходы. Solomon (1987) предложил эвристику последовательной вставки для решения $CVRPTW$. Для тестирования производительности алгоритмов, используемых в проблеме, была сгенерирована задача с различным процентом временных окон и их плотностью.

Глава 1. Задача транспортной маршрутизации

1.1 Математическая постановка задачи

Задачу $CVRPTW$ можно представить в виде ориентированного графа $G = (N, E)$, где $N = \{0, \ldots, n\}$ - множество вершин, E - множество дуг, соединяющих вершины графа.

Введем следующие обозначения:
$C = \{1, \ldots, n\}$ - множество клиентов.
$(i, j) \in E$ - дуга, соединяющая вершину i с вершиной j.
$D = \{1, \ldots, d_n\}$ - множество запросов клиентов.
$t_{i,j}$ - время перевозки по дуге (i, j), с учетом времени обслуживания клиента i,
$c_{i,j}$ - стоимость перевозки груза от клиента i к клиенту j.
V - множество транспортных средств с одинаковой грузоподъемностью Q.
$[a_i, b_i]$ - временной промежуток, в который должен быть обслужен i-ый клиент.
s_i^k - время прибытия транспортного средства k к клиенту i; время отправления любого транспортного средства из депо всегда равно 0.
$x_{i,j}^k = \{0, 1\}$ - переменная, характеризующая направление движения. То есть,

$$x_{i,j}^k = \begin{cases} 1, & \text{если транспортное средство движется из } i \text{ в } j; \\ 0, & \text{если транспортное средство движется из } j \text{ в } i \end{cases}$$

Основные задачи $CVRPTW$: минимизация общего количества задействованных транспортных средств и минимизация общих затрат и расстояний каждого транспортного средства с учётом спроса в узлах, временных окон и грузоподъемности транспорта.

Таким образом, с учётом принятых обозначений, $CVRPTW$ сводится к задаче линейного программирования.

При $a_0 = 0$, $s_{0,k} = 0$ $\forall k$, имеем:

$$\min \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{i,j} x_{i,j}^k \tag{1}$$

при ограничениях

$$\sum_{k \in V} \sum_{j \in N} x_{i,j}^k = 1, \quad i \in \{1, \ldots, n\} \tag{2}$$

$$\sum_{i=1}^{n} d_i \sum_{j \in N} x_{i,j}^k \leq Q, \quad i \in \{1, \ldots, n\} \tag{3}$$

$$\sum_{j \in N} x_{0,j}^k = 1, \quad \forall k \in V \tag{4}$$

$$\sum_{i \in N} x_{i,h}^k - \sum_{j \in N} x_{h,j}^k = 0, \quad \forall h \in \{1, \ldots, n\}, \forall k \in V \tag{5}$$

$$\sum_{i \in N} x_{i,n}^k = 1, \quad \forall k \in V \tag{6}$$

$$x_{i,j}^k (s_i^k + t_{i,j} - s_j^k) \leq 0, \quad \forall i, j \in N, \forall k \in V \tag{7}$$

$$a_i \leq s_i^k \leq b_i, \quad \forall i \in N, \forall k \in V \tag{8}$$

$$x_{i,j}^k \in \{0, 1\}, \quad \forall i, j \in N, \forall k \in V \tag{9}$$

$$\sum_{k \in V} \sum_{j \in N} x_{0,j}^k \leq |V|, \quad \forall j \in N, \forall k \in V \tag{10}$$

Целевая функция 1 минимизирует общие затраты на перевозку товара. Уравнение 2 показывает посещение каждого клиента единожды. Неравенство 3 указывает на ограничение грузоподъемности всех транспортных средств. Уравнения 4, 5 и 6 обозначают отправку из депо, перемещение между клиентами и возвращение в депо соответственно. Неравенство 7 свя-

зывает между собой время отправления из пункта и время прибытия в иной пункт. Двойное неравенство 8 устанавливает временные рамки. Наконец, 9 - это ограничение целостности, а неравенство 10 обозначает ограничение на использование определенного числа транспортных средств.

1.2 Алгоритмы решения

1. Точные алгоритмы

   - Метод ветвей и границ [13]
   - Метод ветвей и отсечений [14]

   Точные алгоритмы всегда дают оптимальные решения. Однако в связи с тем, что задача является NP-сложной, то данный вид алгоритмов является эффективным только при малых размерностях задачи.

2. Эвристические алгоритмы

   - Жадный алгоритм [15]
   - Генетический алгоритм [16]
   - Алгоритм Кернигана-Лина [17]

   Эвристические алгоритмы характерны тем, что основаны на каком-то правиле, правильность которого для всех возможных случаев не доказана, однако в большинстве случаев дают решения, близкие к оптимальному. Такие алгоритмы в процессе работы отсекают огромное количество ненужных (заведомо неоптимальных) решений, что существенно сказывается на их быстродействии решения NP-сложных задач.

3. Мета-эвристические алгоритмы

   - Имитация отжига [18]
   - Поиск с запретами [19]
   - Управляемый локальный поиск [20]

- Алгоритм оптимизации подражанием муравьиной колонии [21]

Отличительной особенностью мета-эвристических алгоритмов перед эвристическими является большая степень эвристики, а также наименьшая вероятность попадать в локальные экстремумы.

Глава 2. Эвристика

2.1 Введение

Не смотря на существенное различие в поведении и характеристиках между эвристиками, они имеют ряд основных факторов, которые являются общими у всех алгоритмов этого типа:

- Инициализация;
- Связь с набором существующих ограничений;
- Отбор кандидатов;
- Критерий принятия кандидатов;
- Критерий остановки.

Целью является минимизация целевой функции затрат, которая имеет ограничение на грузоподъемность и временные окна:

$$F = \sum_{i=1}^{n} t_{d_i} + \sum_{i=0}^{n} r_{d_i,d_{i+1}} + \sum_{i=1}^{n+1} \max(0, s_{d_i} - a_i) + \sum_{k=1}^{m} \max(0, \sum_{i=z_{k-1}+1}^{z_k} p_{d_i} - Q_k),$$

где
$d = \{d_1, \ldots, d_n\}$ - требования спроса узлов и депо;
t_k - время обслуживания в узле k;
$r_{i,j}$ - наименьшее расстояние между i и j;
s_{d_i} - время прибытия к клиенту s_i;
a_i - левая граница временного окна;
p_{d_i} - количество товара в запросе клиента;
z_k - позиция k-ого элемента в строке $d = \{d_1, \ldots, d_n\}$;
Q_k - грузоподъемность транспортного средства.

2.2 Жадный алгоритм

Жадный алгоритм (*Greedy Algorithm*) - это способ выбора наилучшего на данный момент значения на каждом шаге. Здесь имеется в виду

допущение, связанное с выбором локально оптимальных решений на каждом шаге и получением в конце оптимального решения. Алгоритм сильно зависит от размерности задачи и от количества экстремумов в задаче. Так, если существует почти всегда глобальная оптимальность жадного алгоритма, то по быстродействию он намного лучше алгоритмов динамического программирования. Однако не смотря на наивное допущение выбора локального оптимума, в ряде задач алгоритм показывает неплохие решения.

К жадным алгоритмам относят Метод ближайшего соседа. По своей сути, данный метод ничем не отличается от классического жадного алгоритма. Однако в задачах, где граф задается одновременно и через координаты вершин, и через весовые коэффициенты дуг, эти два подхода выступают как самостоятельные алгоритмы.

Работа и реализация алгоритма довольна проста. Изначально имеется пустой маршрут и множество непосещенных точек. Сам алгоритм действует по следующей схеме:

Algorithm 1 Greedy algorithm

```
1:  Nodes ← {1, ..., n}
2:  Path ← {}
3:  CurrentPoint ← Depot
4:  while |Nodes| > 0 do
5:      MinDistance ← ∞
6:      for Point in Nodes do
7:          if Distance(CurrentPoint, Point) ≤ MinDistance then
8:              MinDistance ← Distance
9:              Target ← Point
10:         end if
11:         In Path add Target
12:         From Nodes delete Target
13:         CurrentPoint ← Target
14:         Sum ← Sum + MinDistance
15:     end for
16: end while
```

1. Выбрать начальную точку;

2. Записать эту точку в маршрут и исключить её из множества непосещенных точек;

3. Найти минимальное расстояние от начальной точки до любой точки из множества непосещенных точек. Если таких точек несколько, то выбрать любую;

4. Повторять Шаги 2-3, пока множество непосещенных точек не станет пустым.

Временная сложность алгоритма – $O(n\log(n)+n)$

2.3 Генетический алгоритм

Генетические алгоритмы (*Genetic Algorithms - GA*) представляют собой стохастический подход без производных, основанный на биологических эволюционных процессах, которые предложил Holland [22]. В природе наиболее подходящие особи могут выжить и спариться; следовательно, следующее поколение должно быть более здоровым и крепким, чем предыдущее. Много работы и приложений было сделано по *GA* в часто цитируемой книге Гольберга [23]. Генетические алгоритмы работают с популяцией хромосом, которые представлены некоторыми кодами набора основных параметров. Комплексное исследование подходов к *GA* успешно применяется к *VRP* [24]. Обзор подходов к генетическим алгоритмам для *VRP* был представлен в [25].

Простой и чистый генетический алгоритм может быть определен в следующих шагах:

1. Создать начальную популяцию P хромосом;

2. Оценить приспособленность каждой хромосомы;

3. Выбрать $\frac{P}{2}$ родителей из текущей популяции посредством пропорционального отбора;

4. Случайным образом выбрать двух родителей для создания потомства с помощью оператора кроссовера;

5. Применить оператор мутации для незначительных изменений результатов;

6. Повторить шаги 4-5, пока все родители не будут выбраны;

7. Заменить старую популяцию хромосом на новую;

8. Оценить приспособленность каждой хромосомы в новой популяции;

9. Завершить, если количество поколений достигает некоторой верхней границы; в противном случае, перейти к шагу 3.

Критерии отбора, кроссовер и мутация являются основными операторами, но кроссовер играет жизненно важную роль в генетическом алгоритме.

2.3.1 Критерий отбора

Для алгоритма было реализованно две стратегии: элитарность и избыточный выбор.

Элитарность (*Elitism*) - это способ выбора из популяции только элит, при этом отбрасывая все неэлитные гены. Эта стратегия хороша для более быстрого решения данной задачи, однако она легко попадает в локальные оптимумы. Единственный способ избежать этого - мутация.

Избыточный выбор (*Overselect*) - это способ использовать как элиты, так и неэлиты. Он генерирует $(1-r) \times 100\%$ новых популяций, используя $r \times 100\%$ родителей. Кроме того, при использовании нижних $(1-r) \times 100\%$ родителей генерируется $r \times 100\%$ новой популяции. Итак, если мы устанавливаем $r < 0.5$, это означает, что мы предполагаем, что некоторые из верхних родителей более продуктивны, чем другие нижние родители, и это имеет смысл с точки зрения теории эволюции. В алгоритме используется эта стратегия отбора в качестве метода селекции.

2.3.2 Операторы кроссовера

В литературе было предложено множество операторов кроссовера, и все они имеют большое значение.

Таблица 2: Краткое описание операторов кроссовера для *VRP* [16]

Представление	Операция	Год
Binary	Classical + repair operator	1991
Path	**Partially mapped crossover**	1985
	Order crossover	1985
	Cycle crossover	1987
	Heuristic crossover	1987
	Order based crossover	1991
	Position based crossover	1991
Adjacency	Alternative edge crossover	1985
	Heuristic crossover 1	1985
	Heuristic crossover 2	1989
	Heuristic crossover 3	1987
Ordinal	Classical operators	1985
Matrix	Intersection crossover	1987
	Union crossover	1987

В алгоритме для сравнения использовались 2 выделенных оператора в таблице и один предложенный и описанный ниже.

Циклический кроссовер (*Cycle Crossover - CX*) впервые описан в [26] и буквально заключается в использовании в выборе циклического скрещивания генов, начиная с фиксированного первого гена. Использование этой техники позволяет создать потомство таким образом, что каждый бит со своей позиции поступает от одного из родителей.

Например, рассмотрим двух родителей:

$$P_1 = (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8)$$

$$P_2 = (8 \quad 5 \quad 2 \quad 1 \quad 3 \quad 6 \quad 4 \quad 7)$$

Теперь необходимо решить, как выбрать первый бит для потомка, который будет происходить от первого или от второго родителя. В нашем

примере первый бит потомка должен быть 1 или 8. Пусть он будет равным 1:

$$O_1 = (1 \ \times \ \times \ \times \ \times \ \times \ \times \times)$$

Теперь каждый бит в потомке должен быть взят от одного из его родителей с той же позиции. Это означает, что в дальнейшем у нас нет никакого выбора, поэтому следующим битом, который следует рассматривать, должен быть бит 8, поскольку бит от второго родителя находится под выбранным битом 1. В первом родительском элементе этот бит находится в 8-й позиции; таким образом:

$$O_1 = (1 \ \times \ \times \ \times \ \times \ \times \ \times 8)$$

Это, в свою очередь, подразумевает бит 7, который является битом второго родителя, который находится на 7-й позиции:

$$O_1 = (1 \ \times \ \times \ \times \ \times \ \times \ 7 \quad 8)$$

Это заставляет нас поставить 4 на 4-ую позицию:

$$O_1 = (1 \ \times \ \times 4 \ \times \ \times 7 \quad 8)$$

После этого идет 1, который уже есть в списке. Таким образом, мы завершили цикл и заполнили оставшиеся пустые позиции битами тех позиций, которые находятся во втором родителе:

$$O_1 = (1 \quad 5 \quad 2 \quad 4 \quad 3 \quad 6 \quad 7 \quad 8)$$

Аналогично получаем для второго потомка:

$$O_2 = (8 \quad 2 \quad 3 \quad 1 \quad 5 \quad 6 \quad 4 \quad 7)$$

Частично отображённый кроссовер (*Partially-Mapped Crossover - PMX*) был предложен в [27]. Это способ сопоставить некоторую часть дочернего элемента с соответствующей частью родителя. После выбора двух случайных точек на родителях для создания потомства, часть между точками в строке одного родителя отображается на строку потомка, а оставшаяся информация обменивается.

Рассмотрим, например, двух родителей с одной точкой между 3-им и 4-ым битом и с другой точкой между 6-ым и 7-ым битом:

$$P_1 = (3 \quad 4 \quad 8 \mid 2 \quad 7 \quad 1 \mid 6 \quad 5)$$

$$P_2 = (4 \quad 2 \quad 5 \mid 1 \quad 6 \quad 8 \mid 3 \quad 7)$$

Отображающая часть находится между точками. В этом примере следующая система отображения: $2 \leftrightarrow 1$, $7 \leftrightarrow 6$ и $1 \leftrightarrow 8$. Теперь 2 части отображения переставляются для потомков:

$$O_1 = (\times \quad \times \quad \times \mid 1 \quad 6 \quad 8 \mid \times \quad \times)$$

$$O_2 = (\times \quad \times \quad \times \mid 2 \quad 7 \quad 1 \mid \times \quad \times)$$

Затем можно бесконфликтно заполнить следующие биты:

$$O_1 = (3 \quad 4 \quad \times \mid 1 \quad 6 \quad 8 \mid \times \quad 5)$$

$$O_2 = (4 \ \times \ 5 \mid 2 \quad 7 \quad 1 \mid 3 \quad \times)$$

Следовательно, первое $\times$ в первом потомке - это 8, которое происходит от первого родителя, но 8 уже есть в этом потомке, поэтому проверяется отображение $1 \leftrightarrow 8$ и снова видим 1, существующее в этом потомке. Снова проверяется отображение $2 \leftrightarrow 1$, поэтому 2 занимает первую $\times$. Точно так же второе $\times$ в первом потомке - 6, которое происходит от первого родителя, но 6 существует в этом потомке; также проверяется отображение $7 \leftrightarrow 6$, поэтому 7 занимает вторую $\times$. Таким образом, первый потомок:

$$O_1 = (3 \quad 4 \quad 2 \mid 1 \quad 6 \quad 8 \mid 7 \quad 5)$$

Аналогично для второго потомка:

$$O_2 = (4 \quad 8 \quad 5 \mid 2 \quad 7 \quad 1 \mid 3 \quad 6)$$

В данной работе был предложен к исследованию новый оператор кроссовера, который работает аналогично *CX*, поэтому он будет именоваться как *CX2*. Работа данного кроссовера описывается следующими шагами:

1. Выбрать 2 родителя для скрещивания;
2. Выбрать 1-ый бит второго родителя как 1-ый бит первого потомка;
3. Выбранный бит из шага 2 будет найден в первом родительском элементе и будет выбран точно такой же бит позиции, который находится во втором родительском элементе, и этот бит будет снова найден в первом родительском элементе и, наконец, точно такой же бит позиции, который находится во втором родительском элементе, будет выбранным для 1-го бита второго потомка;
4. Выбранный бит из шага 3 будет найден в первом родительском элементе и выберет точно такой же бит позиции, который находится во

втором родительском элементе, в качестве следующего бита для первого потомка (*Примечание: для первого потомка мы выбираем биты только с одним ходом и два хода для битов второго потомка*);

5. Повторить шаги 3 и 4 до тех пор, пока 1-й бит первого родителя не появится во втором потомке (завершит цикл), и процесс может быть завершен;

6. Если некоторые биты остались, то одни и те же биты в первом родителе и во втором потомке до сих пор (и наоборот) не учитываются у обоих родителей. Для оставшихся битов повторить шаги 2, 3 и 4, чтобы завершить процесс.

   Рассмотрим двух выбранных родителей, как указано в Шаге 1:

$$P_1 = (3 \quad 4 \quad 8 \quad 2 \quad 7 \quad 1 \quad 6 \quad 5)$$

$$P_2 = (4 \quad 2 \quad 5 \quad 1 \quad 6 \quad 8 \quad 3 \quad 7)$$

   На втором Шаге:

$$O_1 = (4 \ \times \ \times \ \times \ \times \ \times \ \times \ \times)$$

Как и при использовании шага 3, который выбрал 4 на шаге 2, где 4 находится во второй позиции в первом родительском элементе, а бит в этой позиции во втором родительском элементе равен 2. Для повторного поиска 2 находится в четвертой позиции в первом родительском элементе, а 1 - в том же позиция во втором родителе, поэтому 1 выбирается для второго потомка следующим образом:

$$O_2 = (1 \ \times \ \times \ \times \ \times \ \times \ \times \ \times)$$

Чтобы выполнить шаг 4, предыдущий бит был 1, и он расположен в 6-й позиции в первом родительском элементе, а в этой позиции бит равен 8 во втором родительском элементе, поэтому:

$$O_1 = (4 \quad 8 \quad \times \quad \times \quad \times \quad \times \quad \times \quad \times)$$

И для следующих двух шагов 5 под 8 и 7 под 5, поэтому:

$$O_2 = (1 \quad 7 \quad \times \quad \times \quad \times \quad \times \quad \times \quad \times)$$

Продолжая аналогичным образом до конца:

$$O_1 = (4 \quad 8 \quad 6 \quad 2 \quad 5 \quad 3 \quad 1 \quad 7)$$

$$O_2 = (1 \quad 7 \quad 4 \quad 8 \quad 6 \quad 2 \quad 5 \quad 3)$$

Псевдокод реализации данного оператора кроссовера представлен в Приложении 1. По умолчанию, вероятность кроссовера равна 0.8.

Оператор мутации представлен весьма просто: выбирается 2 случайных узла у особи и меняются местами. По умолчанию, вероятность мутации составляет 0.05.

Временная сложность алгоритма – $O(g(nm + nm + n))$, где
g - число поколений;
n - размер популяции;
m - размер особей.

2.4 Алгоритм Кернигана-Лина

Алгоритм *2-opt*, о котором упоминалось в разделе 1.2 ([20]), является частным случаем алгоритма *k-opt*, где на каждом шаге k дуг текущего маршрута заменяются k дугами таким образом, что достигается более короткий путь. Другими словами, на каждом шаге получается укороченный

маршрут путём удаления дуг и объединения полученных путей по-новому, возможно, изменяя один или несколько из них в обратном направлении.

Алгоритм *k-opt* основан на концепции *k-optimality*:

> Маршрут называется *k-оптимальным*, если невозможно получить более короткий маршрут, заменив любые k его дуг любым другим набором из k дуг.

Из этого определения очевидно, что любой k-оптимальный маршрут также k'-оптимален для $1 \leq k' \leq k$. Также легко видеть, что маршрут, содержащий n городов, оптимален тогда и только тогда, когда он n-оптимален.

В общем, чем больше значение k, тем больше вероятность того, что финальный маршрут будет оптимальным. К сожалению, количество операций по тестированию всех k-обменов быстро увеличивается по мере возрастания количества вершин графа – в наивной реализации тестирование k-обмена имеет временную сложность $O(n \times k)$. Кроме того, нет нетривиальной верхней границы k, в результате чего наиболее распространенными значениями стали $k = 2$ и $k = 3$. Однако в исследовании [28] использовались значения $k = 4$ и $k = 5$.

Недостатком является то, что k необходимо указывать заранее, ибо трудно понять, что использовать для достижения наилучшего компромисса между временем работы и качеством решения. Lin и Kernighan устранили этот недостаток, представив *variable-opt* алгоритм, который изменяет значение k во время своего выполнения, решая на каждой итерации, каким должно быть это значение. На каждом шаге алгоритм проверяет для всех возрастающих значений k, может ли замена k дуг привести к более короткому маршруту. Иными словами, рассматривается замена r дуг путём выполнения серии тестов, которые должны определить, следует ли рассматривать замену $r + 1$ дугами. Это продолжается до тех пор, пока не будут выполнены некоторые условия остановки.

На каждом шаге алгоритм рассматривает растущий набор потенциального количества обменов дуг, начиная с $r = 2$. Эти замены выбраны таким образом, чтобы новый возможный маршрут мог быть сформирован

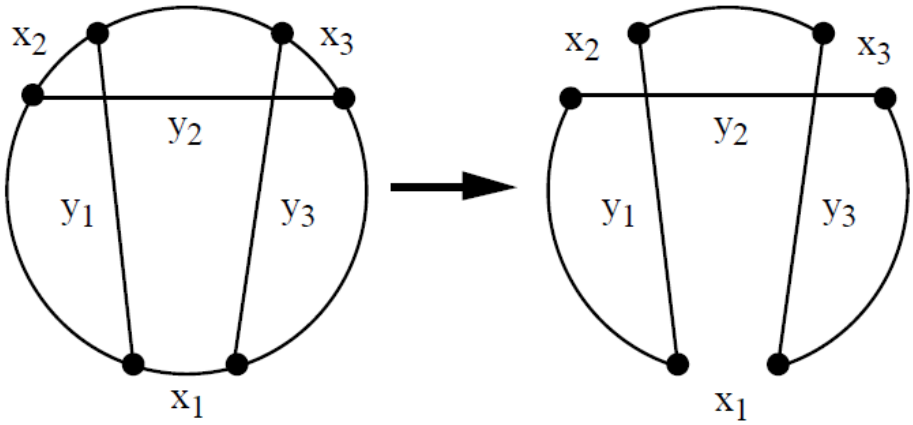

Рис. 1: Шаг в *3-opt* [17]

на любом этапе процесса. Если в ходе исследования удается найти новый, более короткий путь, то текущий маршрут заменяется получившимся.

Алгоритм Кернигана-Лина определяется с точки зрения замены дуг, которые могут конвертировать один маршрут в другой. Алгоритм многократно совершает замену дуг, сокращающих длину текущего пути, пока не будет достигнут маршрут, для которого ни одна замена не даёт улучшения. Этот процесс может повторяться много раз из начальных маршрутов, сгенерированных случайным образом. Рассмотрим подробно алгоритм.

Пусть P - текущий маршрут. На каждой итерации алгоритм пытается найти два набора связей $X = \{x_1, \ldots, x_r\}$, $Y = \{y_1, \ldots, y_r\}$ так, что если дуги X будут удалены из P и заменены дугами из Y, то результатом будет путь лучше. Данная замена дугами называется шагом *r-opt*. Рисунок 1 иллюстрирует шаг для *3-opt* алгоритма.

Наборы X и Y изначально пусты и заполняются поэлементно: на шаге i пара связей x_i и y_i добавляется к X и Y соответственно. Рассматриваемые дуги могут обеспечить достаточной эффективности алгоритма, если будут выполнены следующие критерии:

- Критерий последовательной замены

  x_i и y_i должны иметь одну общую конечную точку, как и y_i и x_{i+1}. Если t_1 обозначает одну из двух точек дуги x_1, то, в общем случае, имеем:

  $x_i = (t_{2i-1}, t_{2i})$

$y_i = (t_{2i}, t_{2i+1})$
$x_{i+1} = (t_{2i+1}, t_{2i+2})$ для $\forall i \geq 1$. См. Рисунок 2

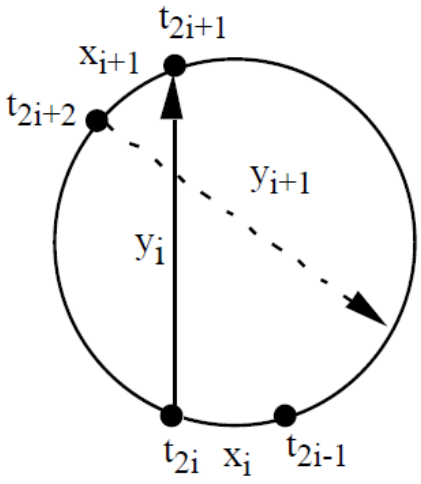

Рис. 2: Пример последовательной замены дуг x_i, y_i и x_{i+1} [17]

Как видно, последовательность $(x_1, y_1, x_2, y_2, \ldots, x_r, y_r)$ представляет собой цепочку смежных дуг. Необходимым (но не достаточным) условием замены дуг X с дугами Y является замкнутость пути, то есть $y_r = (t_{2r}, t_1)$. Такая замена и называется последовательной.

Как правило, улучшение маршрута может быть достигнуто последовательной заменой с помощью подходящей нумерации затронутых дуг, однако это не всегда так.

- Критерий осуществимости
  Требуется, чтобы $x_i = (t_{2i-1}, t_{2i})$ был выбран таким образом, чтобы при соединении t_{2i} с t_1 полученная конфигурация представляла собой полный маршрут. Этот критерий используется для $i \geq 3$ и гарантирует возможность замкнутого пути. Также, он был включен в алгоритм для сокращения времени и упрощения кодирования.

- Критерий положительного выигрыша
  Требуется, чтобы y_i всегда выбирался таким образом, чтобы выигрыш G_i от предложенного набора замены дуг был положительным. Предположим, что $g_i = c(x_i) - c(y_i)$ - выигрыш от замены дуги x_i на

дугу y_i, тогда $G_i = \sum_{k=1}^{i} g_k$. Этот критерий является критерием остановки алгоритма. Требование, чтобы каждая частичная сумма была положительной, кажется слишком ограничивающей, однако это не так. Само условие держится на одном простом факте: если последовательность чисел имеет положительную сумму, то существует циклическая перестановка этих чисел такая, что каждая частичная сумма этих чисел будет положительна. Доказательство этого факта приводится в [15].

- Дизъюнктивный критерий
  Наконец, требуется, чтобы множества дуг X и Y не пересекались. Этот критерий упрощает кодирование, сокращает время работы и также является критерием остановки.

Ниже приводится упрощенная версия алгоритма с учётом всех критериев. Временная сложность алгоритма – $O(p \times n^2 log(n))$, где p - количество итераций в процедуре улучшения.

1. Сгенерировать случайный маршрут P;
2. Пусть $i = 1$. Выбрать t_1;
3. Выбрать $x_1 = (t_1, t_2) \in P$;
4. Выбрать $y_1 = (t_2, t_3) \notin P$ таким образом, что $G_1 > 0$;
   Если это невозможно, то перейти к шагу 12;
5. Пусть $i = i + 1$
6. Выбрать $x_i = (t_{2i-1}, t_{2i}) \in P$ так, что
   - Если t_{2i} соединяется с t_1, то результирующая конфигурация есть маршрут P';
   - $x_i \neq y_s$ для $\forall s < i$;

   Если P' лучше, чем P, то $P \leftarrow P'$. Перейти к шагу 2.

7. Выбрать $y_i = (t_{2i}, t_{2i+1} \notin P)$ так, что:

   - $G_i > 0$;
   - $y_i \neq x_s$ для $\forall s \leq i$;
   - x_{i+1} существует;

   Если такой y_i существует, то перейти к шагу 5.

8. Если есть альтернатива для y_2, то пусть $i = 2$ и перейти к шагу 7;

9. Если есть альтернатива для x_2, то пусть $i = 2$ и перейти к шагу 6;

10. Если есть альтернатива для y_1, то пусть $i = 1$ и перейти к шагу 4;

11. Если есть альтернатива для x_1, то пусть $i = 1$ и перейти к шагу 3;

12. Если есть альтернатива для t_1, то перейти к шагу 2;

13. Стоп (или перейти к шагу 1).

Глава 3. Результаты работы алгоритмов

3.1 Применение алгоритмов

Для проведения эксперимента было рассмотрено три базы данных *Homberger* с тремя различной размерностью в 200, 400 и 600 клиентов, а также с тремя категориями:

- сгруппированные клиенты (*C-type*);
- равномерно распределенные клиенты (*R-type*);
- смесь типов *R* и *C*.

Данные находятся в свободном доступе на сайте *https://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-with-time-windows-instances/*, там же имеются лучшие известные решения (*Best Known Solution – BKS*). Ниже представлен формат задачи.

VEHICLE

NUMBER	CAPACITY
100	1000

CUSTOMER

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME
0	100	100	0	0	3213	0
1	150	26	16	1035	1275	10
2	84	188	17	1891	2131	10
3	83	112	28	20	260	10
4	191	124	7	541	781	10
5	112	174	20	220	460	10
6	146	166	24	1406	1646	10
7	45	95	15	744	984	10

Рис. 3: Фрагмент формата данных из задачи *R2_4_5.txt*

- VEHICLE NUMBER - верхняя граница количества транспортных средств;
- CAPACITY - грузоподъемность каждого транспортного средства;
- CUSTOMER CUST NO. - номер узла, где 0 - депо, а все остальные - клиенты;
- XCOORD. и YCOORD. - координаты узла по X и Y соответственно;

- DEMAND - спрос клиента;
- READY TIME и DUE TIME - временное окно клиента;
- SERVICE TIME - время обслуживание клиента.

Решение производилось на ноутбуке со следующими характеристиками: Процессор Intel Core i5-7200U 2.5GHz, RAM 6GB. Реализация эвристических алгоритмов производилась на высокоуровневом языке программирования Python 3.9.5.

В качестве критерия остановки во всех алгоритмах было выбрано время в 600 секунд; в генетическом алгоритме аналогичным фактором является количество поколений, равное 100. Также основные числовые характеристики генетического алгоритма:

1. Размер популяции = 1000;
2. Вероятность оператора кроссовера = 0.8;
3. Вероятность мутации = 0.05.

В таблицах, приведенных отдельным файлом, представлены округленные результаты тестов: суммарные затраты на пройденное расстояние транспортными средствами и процент точности по сравнению с известным решением, вычисляемый по формуле

$$\% = \frac{best\_known\_value}{experimental\_value} * 100$$

3.2 Выводы

В связи с вычислительной сложностью задачи, получение глобального минимума целевой функции на сетях больших размерностей приводит к характерным для алгоритмов эвристики результатам. Проведённые исследования показали, что подбор правильного алгоритма для решения поставленной проблемы является важнейшим аспектом оптимизации в целом.

Среди рассмотренных алгоритмов наиболее близкие к оптимальным значениям показал алгоритм Кернигана-Лина. Для большинства тестовых примеров алгоритму сполна хватило предоставленного времени в 600 секунд, чтобы выдать хороший результат. Однако существует 2 примера, в которых алгоритм не дал результат. Связанно это, прежде всего, с тем, что при данных ограничениях невозможно построить единый замкнутый маршрут, что является обязательным условием правильной и успешной работы алгоритма. Некоторые значения, приведённые в таблице, показывают значения, меньше известного оптимального. Это объясняется предоставленными данными, которые не были зависимы от алгоритма – изначально имелись уже округлённые до 2-ого знака после запятой значения, когда как экспериментальные результаты получались с большой разрядной сеткой.

Среди оставшихся исследованных алгоритмов сильно выделяется генетический. Не смотря на схожесть полученной прямой зависимости процента решения от размерности задачи, оба алгоритма весьма разнятся во времени реализации. Как и ожидалось, жадный алгоритм решал примеры быстро и некачественно, когда как в среднем процент генетического был немного выше. Связано это с тем, что жадный алгоритм не просчитывает полноценно временные окна, поэтому так велико конечное значение функции. Генетический, в свою очередь, старается приуменьшить количество штрафов в целом, но менее, чем за 100 поколений алгоритм часто попадает в локальный оптимум. Так или иначе, оба алгоритма способны приблизиться к решению, не более чем на 15%.

Заключение

В данной работе исследуется проблема маршрутизации транспортных средств с ограниченной грузоподъемностью и временными окнами. Поскольку известно, что VRP является NP-сложной, для её решения применяются эвристики. Были рассмотрены в достаточном объёме исследования по моделированию и эвристическим алгоритмам VRP:

- Жадный алгоритм (Greedy Algorithm);
- Генетический алгоритм (Genetic Algorithm);
- Алгоритм Кернигана-Лина (Lin-Kernighan Algorithm).

Данные алгоритмы были реализованы на языке Python и рассмотрены на тестовых данных. Полученные округлённые результаты, а также доля от лучших известных значений, позволили выявить эффективный алгоритм среди рассмотренных.

Список литературы

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

[2] P. Toth, D. Vigo. "An overview of vehicle routing problems". In The Vehicle Routing Problem, SIAM, Philadelphia, 2002.

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

[4] Lenstra J.K., Kan R. "Complexity of Vehicle Routing and Scheduling Problem Networks, 11(2), pp.221-227, 1981.

[5] R.Baldacci, M.Battarra, D.Vigo, "Routing a Heterogeneous Fleet of Vehicles In book: "The Vehicle Routing Problem: Latest Advances and New Challenges pp. 3-27, 2008.

[6] Solomon M.M. "Algorithms for the vehicle routing problem with time windows". Transportation Science, 29(2), pp. 156-166, 1995.

[7] Russell R., Igo W. "An assignment routing problem". Networks, No. 9, pp. 1-17, 1979.

[8] Solomon M.M, Desrosiers J. "Time window constrained routing and scheduling problem". Trans. Sci. 22, pp. 1-13, 1988.

[9] Clarke G. and Wright J.W. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". Operations Research. 12(4), pp. 568-581, 1964.

[10] Christofides N., Eilon S. "An Algorithm for the Vehicle-Dispatching Problem". OR, 20(3), pp.309-318, 1969.

[11] Gillett B.E., Miller. L.R. "A Heuristic Algorithm for the Vehicle-Dispatch Problem". Operations Research 22(2), pp. 340-349, 1974.

[12] Cullen FH, Jarvis J. and Ratliff H. D. "Set partitioning based heuristics for interactive routing". Networks, 11(2), pp. 125-143, 1981.

[13] M. L. Fisher, "Optimal Solution of Vehicle Routing Problems Using Minimum K-tree Operations Research 42, 626-642, 1994.

[14] K.M. Wenger. "Generic Cut Generation Methods for Routing Problem Ph.D Dissertation, Institute of Computer Science, University of Heidelberg, Germany, ISBN 3-8322-2545-5, 2003.

[15] Lin, S., Kernighan, B.W. "An effective Heuristic Algorithm for the TSP Operations Research, No. 21, pp. 498–516, 1973.

[16] Abid Hussain, Yousaf Shad Muhammad, M. Nauman Sajid, Ijaz Hussain, Alaa Mohamd Shoukry, Showkat Gani, "Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator Computational Intelligence and Neuroscience, vol. 2017, Article ID 7430125, 7 pages, 2017.

[17] Keld Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic European Journal of Operational Research, Vol. 126, Issue 1, pp. 106-130, October 2000.

[18] Mahmudy, Wayan."Improved simulated annealing for optimization of vehicle routing problem with time windows (VRPTW)". Kursor. 7. 109-116, 2014.

[19] S. Parker."A Tabu Search Algorithm for the Vehicle Routing Problem with Time Windows". November 27, 2015.

[20] Arnold, Florian Sörensen, Kenneth. "Knowledge-guided local search for the Vehicle Routing Problem". Computers Operations Research, 2019.

[21] Silalahi B.P., Fathiah N., Supriyo P.T. "Use of Ant Colony Optimization Algorithm for Determining Traveling Salesman Problem Routes". Jurnal Matematika MANTIK, 5(2), 100-111, 2019.

[22] J. H. Holland, "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence University of Michigan Press, Oxford, UK, 1975.

[23] D. E. Golberg, "Genetic algorithms in search, optimization and machine learning Addison-Wesley Publishing Company, 1989.

[24] M. Gen and R. Cheng, "Genetic Algorithms and Engineering Design John Wiley & Sons, London, UK, 1997.

[25] J.-Y. Potvin, "Genetic algorithms for the traveling salesman problem,"Annals of Operations Research, vol. 63, pp. 339–370, 1996.

[26] I. M. Oliver, D. J. d. Smith, andR. C. J.Holland, "Study of permutation crossover operators on the traveling salesman problem in Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms, July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA, USA, 1987.

[27] D. Goldberg and R. Lingle, "Alleles, Loci and the Traveling Salesman Problem in Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications, vol.1985, pp. 154–159, Los Angeles, USA.

[28] Christofides, N., Eilon, S. "Algorithms for Large-Scale Travelling Salesman Problems". Operational Research Quarterly (1970-1977), 23(4), pp.511-518, 1972.

[29] Xiaoyan Li, "Capacitated Vehicle Routing Problem with Time Windows: A Case Study on Pickup of Dietary Products in Nonprofit Organization A Thesis Presented in Partial Fulfillment of the Requirements for the Degree Master of Science, Arizona State University, 2015.

Приложения

Приложение 1

Псевдокод оператора кроссовера для генетического алгоритма [16]

Algorithm Genetic Algorithm

1: $N \leftarrow No.ofEdges$
2: $M \leftarrow PopulationSize$
3: $G \leftarrow No.ofGenerations$
4: $A \leftarrow RandomPopulation$
5: **for** $1 \leq i \leq N$ **do**
6: **for** $1 \leq j \leq N$ **do**
7: $Distance \leftarrow sqrt((x_1 - x_2)^2 + (y_1 - y_2)^2)$
8: **end for**
9: **end for**
10: **for** $1 \leq i \leq M$ **do**
11: $R(i, :) \leftarrow RandomPermutation(N)$
12: **end for**
13: **for** $1 \leq generation \leq G$ **do**
14: **for** $1 \leq i \leq M$ **do**
15: $Sum1 \leftarrow Distance(R(1, N), R(i, N))$
16: **for** $1 \leq j \leq N - 1$ **do**
17: $Sum2 \leftarrow Distance(R(i, j), R(i, j + 1)) + Sum2$
18: **end for**
19: $R(N + 1) \leftarrow Sum1 + Sum2$
20: **end for**
21: $R \leftarrow Sort(R)$
22: $B \leftarrow 0.8 * M$
23: $C \leftarrow 0.1 * M$
24: $Length \leftarrow Length(B)/2$
25: **for** $1 \leq i \leq Length$ **do**
26: $C1 \leftarrow zeros(N)$
27: $C2 \leftarrow zeros(N)$
28: $P1 \leftarrow R(B(2 * i - 1), 1 : N)$
29: $P2 \leftarrow R(B(2 * i), 1 : N)$

Algorithm Genetic Algorithm (Continuation)

30: $St1 \leftarrow 0$
31: $St2 \leftarrow 0$
32: $Where(Length(C1) \ = Length(P1))$
33: **end for**
34: $C1(St1) \leftarrow P2(St1)$
35: **while** $St1 < N$ **do**
36: $Ind1 \leftarrow find(P1 == C1(St1))$
37: $Val1 \leftarrow P2(Ind1)$
38: $C2(St2) \leftarrow Val1$
39: $St1 \leftarrow St1 + 1$
40: $Ind2 \leftarrow find(P2 == C2(St2))$
41: $Val2 \leftarrow P1(Ind2)$
42: $C1(St1) \leftarrow Val2$
43: $St2 \leftarrow St2 + 1$
44: **end while**
45: **end for**
46: $R(M + 2 * i - 1, 1 : N) = C1$
47: $R(M + 2 * i, 1 : N) = C1$
48: **for** $1 \leq i \leq (M + 0.8 * M)$ **do**
49: $Sum1 \leftarrow Distance(R(1), (2))$
50: **for** $1 \leq j \leq N - 1$ **do**
51: $Sum2 \leftarrow Distance(R(j), (R(j + 1)) + Sum1)$
52: **end for**
53: $R(N + 1) \leftarrow (Sum1 + Sum2)$
54: $R \leftarrow Sort(R)$
55: $R \leftarrow R(1 : M)$
56: $Z \leftarrow min(R)$
57: **end for**