Санкт-Петербургский политехнический университет
Петра Великого (СПбПУ)

Физико-механический институт

Курсовая работа
по теме

Задача коммивояжера с временными ограничениями

дисциплина "Методы оптимизации"

Направление 03.01.02
Прикладная матетатика и информатика

Студент
гр. 5030102/90201 _____ И. В. Попов
муравьиный алгоритм;
постановка и формализация задачи

Студент
гр. 5030102/90201 _____ А. С. Струков
метод ближайшего соседа;
исследование метода

Преподаватель
доцент, к.ф.–м.н. _____ Е. А. Родионова

Санкт-Петербург 2022

Содержание

1 Введение

Задача коммивояжера занимает умы специалистов уже не один десяток лет простотой своей формулировки, конечностью множества допустимых решений, наглядностью, и, в тоже время, сложностью решения, поскольку данная задача относится к классу NP-полных задач, то есть не существует алгоритма, который бы находил точное решение за полиномиальное время.

В рамках курсовой работы рассматривается модификация задача ассиметричного коммивояжера с временными окнами. Суть данной модификации в том, что каждому городу, который должен посетить коммивояжер, ставится в соответствие временной промежуток, в который этот город можно посетить.

Обычная задача коммивояжера на практике, может, и не представляет большого интереса, а скорее интересна как чисто математическая проблема, куда важнее считается обобщение задачи коммивояжера для транспорта и логистики, когда несколько транспортных средств ограниченной грузоподъемности должны обслуживать клиентов, посещая их в заданные промежутки времени.

Таким образом, ставятся следующие цели:

- Поставить и формализовать задачу коммивояжера с временными окнами.
- Изучить точный метод решения задачи коммивояжера - метод полного перебора и его модификации.
- Исследовать метод муравьиной колонии.
- Программно реализовать данные методы.
- Сравнить полученные алгоритмы.

2 Постановка задачи

Рассматривается следующая формулировка задачи коммивояжера. Есть N городов, соединённых между собой односторонними дорогами. Коммивояжер выезжает из начального города, он должен посетить остальные N-1 городов и закончить свой путь в некотором заранее определенном городе(или вернуться обратно в стартовый город). Повторное посещение городов запрещено. Дополнительно для каждого города указано временное окно, в которое возможно его посетить. Требуется найти оптимальный маршрут, проходящий через каждый город.

3 Формализация

Ассиметричную задачу коммивояжера с времеными ограничениями формализуем следующим образом. Введем конечный граф

$$G(V, E),$$

где $V = N; N = \{1, \ldots, n\}$ - множество вершин(городов); $E = N \times N$ - множество ребер(дорог, соединяющих города).
Обозначим $s \in N$ - начальная вершина(отправная точка), $f \in N$ - конечная вершина.
Пусть:
$D_{ij} =< x_i, x_j >$ - длина ребра, соединяющего вершины x_i и x_j; $x_i, x_j \in N$;

$$X_{ij} = \begin{cases} 1 & \text{ребро } D_{ij} \text{ входит в оптимальный маршрут,} \\ 0 & \text{ребро } D_{ij} \text{ не входит в оптимальный маршрут;} \end{cases}$$

Пусть $[a_i, b_i]$ - временной интервал, в который можно посетить вершину $x_i \in N \setminus \{s, f\}$, где a_i - начало временного интервала, в который допускается посещение данной вершины, b_i - его окончание. Соответственно, a_s - время выезда коммивояжёра из начальной вершины s, b_f - время прибытия в вершину f.

Поскольку допускается ожидание начала временного интервала, то пусть w_i - время ожидания в вершине x_i, тогда t_i - время прибытия коммивояжёра в вершину $x_i \in N \setminus \{s, f\}$. Соответственно, t_s - время начала маршрута, t_f - время прибытия в вершину f.
$T_{ij} = (t_j + w_j) - (t_i + w_i)$ - разница между временем начала обслуживания вершины x_i и временем начала обслуживания x_j, где $t_j = t_i + w_i + D_{ij}$.
$A_{ij} = b_j - (t_i + w_i + D_{ij})$ - время, оставшееся до последней возможности посетить вершину x_j.
При этом каждому ребру соответствует стоимость $c_{ij} = l_1 T_{ij} + l_2 A_{ij} + l_3 D_{ij}$,
где $l_1 + l_2 + l_3 = 1, l_1 \geq 0, l_2 \geq 0, l_3 \geq 0$ - коэффициенты, позволяющие влиять на стоимость пути.
Тогда получим целевую функцию:

$$\sum_{i \in N j \in N} c_{ij} X_{ij} \rightarrow min$$

Теперь рассмотрим множество ограничений, накладываемых на переменные:

1. Условия неотрицательности и целочисленности:

$$D_{ij} \geq 0 \forall i, j \in N$$
$$X_{ij} \geq 0 \forall i, j \in N; X_{ij} \in \{0, 1\}$$

2. Условие на посещение всех вершин в соответствие с временными рамками:

$$t'_i + w_i + D_{ij} \leq b_j, t'_i = max\{a_i, t_i\}, i \neq j$$

3. Условие на временные окна:

$$0 \leq a_i \leq t_i + w_i \leq b_i, \forall i \in N$$

4. Условие на то, что коммивояжёр посетит каждую вершину только один раз:

$$\sum_{j \in N \setminus \{f\}} X_{ij} = 1, \forall i \in N$$
$$\sum_{i \in N \setminus \{s\}} X_{ij} = 1, \forall j \in N$$

5. Условие на то, что коммивояжёр может выехать из вершины, только если он до этого в нее прибыл:

$$\sum_{i \in N \setminus \{f\}} X_{ij} - \sum_{i \in N \setminus \{s\}} X_{ji} = 0, \quad \forall j \in N$$

4 Решение задачи

4.1 Алгоритм ближайшего соседа

4.1.1 Описание

Модификация полного перебора. Осуществляет поиск одного решения.

Метод ближайшего соседа пытается выбрать самый близкий к текущему непосещенный город. Близость может определяться по различным критериям:

- расстояние между городами
- времена открытий
- времена закрытий

Если же город не удалось посетить из текущего города, меняется текущий город, путем возвращения в предыдущий. Начинается цикл поиска нового города.

4.1.2 Краткий алгоритм построения пути

Строим путь пока возможно

Если из города существуют дороги лишь в посещенные города, запрещаем путь из предыдущего города к текущему, разрешаем все запрещенные пути из этого города, уезжаем в предыдущий город

Снова пытаемся построить путь

4.1.3 Подробный алгоритм поиска пути

1. добавляется начальный город (фиксированный)
2. выбирается ближайшая дорога, ведущая из текущего города в еще непосещенный

   если существует дорога в непосещенный город
   - переходим в этот город
   - убираем найденную дорогу из числа возможных
   - повторяем п.2 с этим городом в качестве текущего

   если не существует непосещенной дороги из текущего города

   если длина пути равна количеству городов
   путь найден!

   если длина пути не равне количеству городов
   * предыдущий город становится текущим
   * восстанавливаем ограниченные ранее пути из этого города
   * запрещаем путь из предыдущего города в текущий
   * убираем текущий город из посещенных городов

4.2 Муравьиный алгоритм

4.2.1 Описание

Муравьиный алгоритм можно классифицировать как вероятностный, то есть он дает только приближенное решение, не гарантируя его оптимальности.

Муравьиный алгоритм, как можно понять по его названию, основан на модели поведения муравьев, ищущих пути от источника пищи до колонии. Проходя по такому пути, муравьи откладывают ферамоны. Другие же сородичи, находя тропы с феромонами, вероятнее всего ими воспользуются, тем самым укрепляя феромонную тропу. Такое неявное взаимодействие между индивидами, в данном случае муравьями, называется стигмергией. При этом, чем меньше путь до источника пищи, тем меньше времени понадобится на него муравьям - следовательно, тем быстрее оставленные на нем следы будут заметны. В итоге, чем больше муравьев проходит по определенному пути, и чем этот путь короче, тем он становится более привлекательным для остальных муравьев.

4.2.2 Формализация

Приведем краткое описание описание работы алгоритма.

1. Инициализация переменных

   Задается переменная m - количество муравьев, участвующих в построении пути.

   Понятно, что раз муравьи ещё ни разу не прошли по пути, то и феромону неоткуда взяться на ребрах, поэтому перед началом поиска оптимального пути количество феромона рассчитывается для каждого ребра исходя из примерной оценки протяженности пути при использовании эвристики ближайшего соседа.

   $\tau_{ij}(0) = \frac{1}{(n * L_N)}$ - формула вычисления количества феромонов, где $\tau_{ij}(0)$ - количество феромона, отложенного на ребре D_{ij} в начальный момент времени;
   n - количество городов;
   L_N - длина маршрута, найденная при использовании эвристики ближайшего соседа.

   Стоит сказать пару слов об эвристике ближайшего соседа. Она выглядит примерно следующим образом:

   1. Выбрать произвольную вершину x_i как текущую.
   2. Найти кратчайшее ребро, соединяющее вершину x_i и любую непосещенную вершину x_j.
   3. Установить x_j как x_i, т.е. перейти в данную вершину и пометить её как посещенную.
   4. Если не осталось непосещенных вершин, то алгоритм окончен, иначе пункт 2.

2. Построение пути

   На этом шаге происходит построение пути каждым муравьем, которого мы помещаем в случайный город, с учетом ограничений и эвристик.

   В цикле по всем агентам от $k = 1$ до $k = m$
   Если ещё есть непосещенные вершины, то:

2.1 Локальная эвристика

Подсчитывем две локальные эвристики f_{ij} и w_{ij}, они вычисляются каждый раз, когда агенту требуется выбрать следующую непосещенную вершину в своем пути.

Эвристика f_{ij} определяется следующим образом. Для k-го агента, находящегося в вершине x_i время за которое он прибудет в вершину x_j будет равно $t_j = t'_i + w_i + D_{ij}$, где $t'_i = max\{a_i, t_i\}$. Зададим оставшееся время до закрытия верщины x_j через переменную $F_{ij} = b_j - t_j$.

Смысл данной эвристики в том, что агенту желательнее посетить города, у которых ближе срок закрытия временного окна посещения, это позволить избежать опоздания в города с более поздними временными интервалами. Чтобы обеспечить это, предполагается выполнение следующего отношения:

$$f_{ij} = \begin{cases} \dfrac{1}{1 + \exp(\delta(F_{ij} - \sigma))}, & F_{ij} \geqslant 0 \\ 0, & F_{ij} < 0 \end{cases}$$

где δ - контролирует кривизну функции, значение присваивается на основе эксперементальных данных;
σ - точка перегиба, определяется как среднее значение всех $F_{ij} \geq 0$.

Эвристика w_{ij} нужна для уменьшения времени ожидания открытия временного интервала. Определим время ожидания в вершине x_j как $W_{ij} = a_j - t_j$. Получим следующее отношение:

$$w_{ij} = \begin{cases} \dfrac{1}{1 + \exp(\lambda(W_{ij} - u))}, & W_{ij} > 0 \\ 1, & W_{ij} \leqslant 0 \end{cases}$$

где λ - контролирует кривизну функции, значение присваивается на основе эксперементальных данных;
u - точка перегиба, определяется как среднее значение всех $W_{ij} > 0$.

В итоге вершина с меньшим временем ожидания и меньшим оставшимся временем до закрытия временного интервала с высокой долей вероятности будет выбранной следующей для посещения муравьем.

2.2 Перемещение

Выбирается следующая из непосещенных вершин для перехода. Для выбора следующей вершины существуют две отдельные стратегии: эскплуатация и эксплорация.

При эксплорации выбирается вершина, для которой произведение $\tau_{ij}(t) * f_{ij}^{\beta} * w_{ij}^{\gamma}$ максимально по сравнению с другими непосещенными вершинами, где β, γ - коэффициенты эвристики, $\beta + \gamma = 1$ Чем больше значение коэффициента, тем больший вклад мы ему придаем, и наоборот.

При эксплуатации пути всегда выбирается случайная непосещенная вершина, для которой произведение $\tau_{ij}(t) * f_{ij}^{\beta} * w_{ij}^{\gamma}$ положительно.

Стратегия, которая будет использована для перехода, выбирается случайно за счет генерации случайного числа q, равномерно распределенного в интервале [0,1], и числа $q_0 (0 \leq$

$q_0 \leq 1$), которое определяется пользователем и отвечает за вероятность эксплуатации перехода.

Переходим в выбранную вершину и отмечаем её как посещенную.

2.3 Подсчет стоимости пути

Переходя по каждому ребру, обновляем переменную, отвечающую за время в пути данного агента - L^k; Для произвольной вершины $L^k = t'_i$.

Также, пройдя по всему маршруту, расчитывется переменная, отвечающая за длину пути данного агента - D^k.

3. Запомнить лучший маршрут из найденных

Если полученное локальное решение лучше глобального, то сохраняем его и обновляем глобальное решение.

4. Заново отмечаем для всех агентов все вершины как непосещенные.

5. Применяем локальное правило обновления феромона

После успешного построения маршрута, уровень феромона обновляется на каждом ребре, пройденном агентом в течение пути. Данный шаг имитирует прокладку феромонных следов, оставляемых муравьями в природе на пути к пище. Наиболее успешные маршруты получают больше всего феромонов, поскольку преодолеваются муравьями быстрее.

Математически это записывется так:
$\tau_{ij}(t) = (1 - w) * \tau_{ij}(t-1) + w * \Delta\tau_{ij}(t)$,
где $0 < w < 1$ - параметр локального испарения феромона, задаваемый пользователем;
$\Delta\tau_{ij}(t)$ - переменная, показывающая насколько увеличится количество феромона на ребре D_{ij} в промежуток времени [(t-1),t], представляет собой сумму всего феромона, отложенного каждым агентом, прошедшим по данному ребру в данный период времени, т.е.: $\Delta\tau_{ij}(t) = \sum_k \Delta\tau^k_{ij}(t)$
$\Delta\tau^k_{ij}(t)$ - это количество феромона, отложенного k-ым агентом на ребре D_{ij} в промежуток времени [(t-1),t], и вычисляется оно по следующей формуле:

$$\Delta\tau^k_{ij}(t) = \begin{cases} \dfrac{1}{L_k(t)}, & (i,j) \in T_k \\ 0 \quad , & (i,j) \notin T_k \end{cases}$$

где T_k - маршрут, завершенный k-ым агентом.
L_k - длина маршрута k-го агента.

6. Применяем глобальное правило обновление феромона

На этом шаге феромон испаряется с ребер. Данный шаг необходим и исполняется после каждой итерации алгоритма, когда каждый агент успешно выстроил маршрут и применил правило локального обновления феромона.

Математически это выглядит так:
$\tau_{ij}(t) = (1 - p) * \tau_{ij}(t-1) + p * \Delta\tau_{ij}(t)$,

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \dfrac{1}{L_k^*}, & (i,j) \in T_k \\ 0, & (i,j) \notin T_k \end{cases}$$

где $0 < p < 1$ - коэффициент глобального испарения феромона;
L_k^* - время прохождения лучшего глобального решения T_k с начала работы алгоритма.
Глобальное правило обновления феромона необходимо для большего накапливания феромона на ребрах, входящих в лучший маршрут.

7. Проверка критерия остановки

   Для данного алгоритма критерием остановки можно задать количество итераций выполнения шагов 2-6, другими словами количество поколений муравьев, которые будут искать и прокладывать оптимальный маршрут.

   Другой способ - это останавливать работу алгоритма после определенного числа итераций, если глобальное решение долго не менялось.

   Если критерий остановки достигнут, то работа алгоритма завершена, иначе пункт 2.

5 Пример

5.1 Постановка задачи

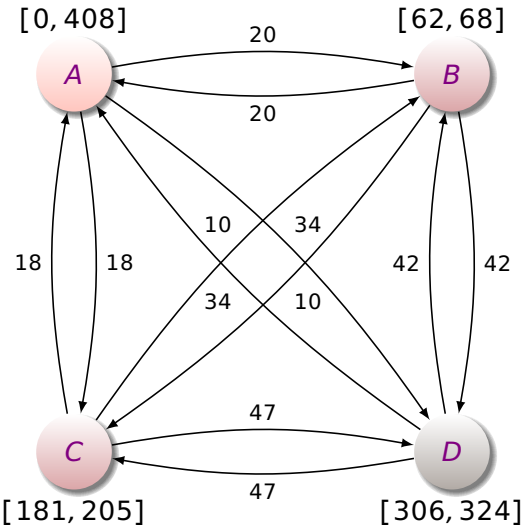
Задана система дорог между городами A, B, C, D, которая определяется следующией матрицей весов:

$$\begin{pmatrix} 0 & 20 & 18 & 34 \\ 20 & 0 & 10 & 42 \\ 18 & 10 & 0 & 47 \\ 34 & 42 & 47 & 0 \end{pmatrix}$$

Время работы городов следующее:

Время открытия	закрытия
0	408
62	68
181	205
306	324

Пусть, стартовая вершина — A, конечная — D.
Объединяя известную информацию о задаче, построим граф, указав для каждой вершины ее временное окно, а для каждого ребра — его вес.



5.2 Решение методом поиска в глубину с возвратом

Отсортируем города по времени закрытия - критерий близости городов: $B \to C \to D \to A$.

Поскольку A — начальный город, начальное время равно времени открытия данного города $time = open\_time[A] = 0$.

Попробуем посетить ближайший к текущему город - B: $time + |AB| <= close\_time[B]$? Да, передвигаемся в город B. $time = max\{time + |AB|, open\_time[B]\}$
Попробуем посетить ближайший к текущему город - C: $time + |BC| <= close\_time[C]$? Да, передвигаемся в город C. $time = max\{time + |BC|, open\_time[C]\}$
Попробуем посетить ближайший к текущему город - D: $time + |CD| <= close\_time[D]$? Да, передвигаемся в город D. $time = max\{time + |CD|, open\_time[D]\}$

Можно ли вернуться в начальный город из текущего: $time + |DA| <= close\_time[A]$? Да, передвигаемся в начальный город A. $time = max\{time + |DA|, open\_time[A]\}$.

Путь построен! $A \to B \to C \to D \to A$

5.3 Решение с помощью муравьиного алгоритма

Зададим параметры для алгоритма: $m = 2; \gamma = \beta = 0.5; \sigma = \alpha = 0.01; q_0 = 0.9$
Определим $\tau_{ij}(0) = \frac{1}{(n\&L_N)}$. По эвристике ближайшего соседа получаем следующий путь $x_1 - x_3 - x_2 - x_4 - x_1$, тогда значение феромона для каждого ребра в графе:

$$\tau_{ij}(0) = \frac{1}{4*(18+10+42+34)} = 0.02$$

Поместим первого муравья в вершину вершину x_1, второго в вершину x_2, и покажем расчеты для первого муравья, для второго можно посчитать по аналогии.
Начинаем строить кратчайший маршрут для первого муравья:
Есть три вершины для посещения x_2, x_3, x_4, для каждой из них:
$t'_1 = a_0$

1. x_2
   (a) Проверить условие $t'_1 + D_{12} \le b_2$; $0 + 20 \le 68$; $t_2 = 20$
   (b) $F_{12} = b_2 - t_2 = 68 - 20 = 48$; $f_{12} = \frac{1}{1+e^{(0.01*(48-175))}} = 0.78$
   (c) $W_{12} = a_2 - t_2 = 62 - 20 = 42$; $w_{12} = \frac{1}{1+e^{(0.01*(42-159))}} = 0.763$
   (d) $\tau_{12}(0) * f_{12}^{0.5} * w_{12}^{0.5} = 0.02 * (0.78 * 0.763)^{0.5} = 0.0154$
2. x_3
   (a) Проверить условие $t'_1 + D_{13} \le b_3$; $0 + 18 \le 205$; $t_3 = 20$
   (b) $F_{13} = b_3 - t_3 = 205 - 18 = 187$; $f_{13} = \frac{1}{1+e^{(0.01*(187-175))}} = 0.47$
   (c) $W_{13} = a_3 - t_3 = 181 - 18 = 163$; $w_{13} = \frac{1}{1+e^{(0.01*(169-159))}} = 0.49$
   (d) $\tau_{13}(0) * f_{13}^{0.5} * w_{13}^{0.5} = 0.02 * (0.47 * 0.49)^{0.5} = 0.0096$
3. x_4
   (a) Проверить условие $t'_1 + D_{14} \le b_4$; $0 + 34 \le 324$; $t_4 = 34$
   (b) $F_{14} = b_4 - t_4 = 324 - 34 = 290$; $f_{14} = \frac{1}{1+e^{(0.01*(290-175))}} = 0.24$
   (c) $W_{14} = a_4 - t_4 = 306 - 34 = 272$; $w_{14} = \frac{1}{1+e^{(0.01*(272-159))}} = 0.24$
   (d) $\tau_{14}(0) * f_{14}^{0.5} * w_{14}^{0.5} = 0.02 * (0.24 * 0.24)^{0.5} = 0.0048$

Везде выше использовались $\delta = \frac{48+187+290}{3} = 175$, $\lambda = \frac{42+163+272}{3} = 159$.
Генерируем случайное число $q = 0.7$
Выбираем следующую вершину через максимальное значение произведения $\tau_{ij}(t) * f_{ij}^{\beta} * w_{ij}^{\gamma}$. Максимальное значение соответствует вершине x_2.
$t'_{12} = 62$;
Есть две вершины для посещения x_3, x_4, для каждой из них:

1. x_3
   (a) Проверить условие $t'_{12} + D_{23} \leq b_3$; $62 + 10 \leq 205$; $t_3 = 72$
   (b) $F_{23} = b_3 - t_3 = 205 - 72 = 133$; $f_{13} = \frac{1}{1+e^{(0.01*(205-175))}} = 0.607$
   (c) $W_{23} = a_3 - t_3 = 181 - 18 = 163$; $w_{13} = \frac{1}{1+e^{(0.01*(163-159))}} = 0.61$
   (d) $\tau_{23}(0) * f_{23}^{0.5} * w_{23}^{0.5} = 0.02 * (0.607 * 0.61)^{0.5} = 0.0122$
2. x_4
   (a) Проверить условие $t'_{12} + D_{24} \leq b_4$; $62 + 42 \leq 324$; $t_4 = 104$
   (b) $F_{24} = b_4 - t_4 = 220$; $f_{24} = \frac{1}{1+e^{(0.01*(220-175))}} = 0.3929$
   (c) $W_{24} = a_4 - t_4 = 306 - 228 = 78$; $w_{14} = \frac{1}{1+e^{(0.01*(78-159))}} = 0.3858$
   (d) $\tau_{24}(0) * f_{14}^{0.5} * w_{14}^{0.5} = 0.02 * (0.3929 * 0.3858)^{0.5} = 0.0078$

Везде выше использовались $\delta = \frac{133+220}{2} = 176.5$, $\lambda = \frac{109-220}{2} = 155.59$.
Генерируем случайное число $q = 0.1$
Выбираем следующую вершину через максимальное значение произведения $\tau_{ij}(t) * f_{ij}^{\beta} * w_{ij}^{\gamma}$. Максимальное значение соответствует вершине x_3.
$t'_{23} = 181$;
Есть одно вершина для посещения x_4, для каждой из них:

1. x_4
   (a) Проверить условие $t'_{23} + D_{34} \leq b_4$; $181 + 47 \leq 324$; $t_4 = 288$
   (b) $F_{34} = b_4 - t_4 = 324 - 233 = 96$; $\delta = F$; $f_{34} = 0.5$
   (c) $W_{34} = a_4 - t_4 = 306 - 233 = 78$; $\lambda = W$; $w_{34} = 0.5$
   (d) $\tau_{34}(0) * f_{34}^{0.5} * w_{34}^{0.5} = 0.01$
   (e) $L^k = t'_{34} = 306$

Вернемся к вершине x_1, чтобы путь выглядел $x_1 - x_2 - x_3 - x_4 - x_1$. Время продолжительности Маршрута L^k =306 + 34 = 340; расстояние D^k =20 + 10 + 47 + 34 = 110
Далее алгоритм работает в своем обычном режиме до точки остановки.

6 Исследования

6.1 Метод ближайшего соседа с возвратами

В ходе данной курсовой работы был исследован метод ближайших соседей с возвратами. Объектом исследования выступает скорость нахождения любого, не обязательно оптимального, пути.

Были проведены тесты алгоритма с различным критерием "близости":

- расстояние между городами
- время открытия
- время закрытия

Результаты представлены в виде таблицы (таб. 1). По этим данным видно, что при выборе ближайшего соседа по времени закрытия, алгоритм выдает лучшие времена работы. Из этого можно сделать вывод, что времена закрытия имеют больший вес в построении решения задачи, нежели расстояния между городами и времена открытия.

Стоит также отметить, что алгоритм может выродиться в полный перебор, если, например, первый посещенный город был выбран ошибочно. Отчасти это происходит из-за следующего: если непосещенный город нельзя посетить из данного города, алгоритм будет пытаться посетить другой город из этого города. Чтобы сменить данный, несостоявшийся как город торговли, город, алгоритм будет перебирать все непосещенные города, что в корне неверно!

Для всех дорог между городами выполнено правило треугольника: длина пути между двумя городами не превосходит длину пути между этими двумя городами, проходящего через любой другой город. Алгоритм игнорирует данное обстоятельство, что существенно повышает его алгоритмическую сложность. Была предпринята попытка заложить вышеизложенное в алгоритм, но, к сожалению, она не увенчалась успехом.

filename	cities #	path	open times	close times	filename	cities #	path	open times	close times
rc_206.1.txt	4	0.000115	0.000095	0.000090	n20w80.001.txt	21	0.430352	0.010380	0.000159
rc_207.4.txt	6	0.001484	0.001440	0.001372	n20w80.002.txt	21	0.116918	>5	0.000150
rbg010a.tw	11	0.000249	0.000081	0.000060	rbg020a.tw	21	>5	>5	0.000154
rc_202.2.txt	14	0.004483	0.003691	0.000079	rc_204.3.txt	24	>5	3.948723	0.000180
rc_205.1.txt	14	0.253974	0.000171	0.000082	rbg027a.tw	28	>5	>5	0.000209
rc_203.4.txt	15	>5	0.330863	0.000108	rbg031a.tw	32	>5	0.000248	0.000209
rbg017.2.tw	16	>5	0.000108	0.000090	rbg033a.tw	34	>5	0.000256	0.000236
rbg017.tw	16	>5	0.000109	0.000090	rbg034a.tw	35	>5	0.000259	0.000239
rbg016a.tw	17	0.283644	0.000113	0.000116	rbg035a.2.tw	36	>5	>5	0.000334
rbg016b.tw	17	0.251956	0.000134	0.000121	rbg035a.tw	36	>5	0.000270	0.000287
rbg017a.tw	18	0.000311	>5	0.000126	rbg038a.tw	39	>5	0.000293	0.000277
rc_203.1.txt	19	>5	>5	0.084101	n40w20.003.txt	41	>5	>5	0.000363
rbg019a.tw	20	0.665621	0.000143	0.000120	rbg040a.tw	41	>5	0.000322	0.000299
rbg019b.tw	20	>5	0.000131	0.000115	rbg041a.tw	42	>5	0.000356	0.000310
rbg019c.tw	20	>5	>5	0.000140	rbg042a.tw	43	>5	0.000352	0.000347
rbg019d.tw	20	1.120524	0.000133	0.000115	rbg048a.tw	49	>5	>5	0.000439
rbg021.2.tw	20	>5	>5	0.000142	rbg049a.tw	50	>5	>5	0.000493
rbg021.3.tw	20	>5	>5	0.000146	rbg050a.tw	51	>5	0.000465	0.000430
rbg021.4.tw	20	>5	>5	0.000146	rbg050b.tw	51	>5	>5	0.000480
rbg021.5.tw	20	>5	>5	0.000148	rbg050c.tw	51	>5	>5	0.000480
rbg021.6.tw	20	>5	0.000146	0.000124	rbg055a.tw	56	>5	0.000542	0.000561
rbg021.7.tw	20	>5	0.000140	0.000118	rbg067a.tw	68	>5	0.000651	0.000644
rbg021.8.tw	20	0.000221	0.000117	0.000116	rbg092a.tw	93	>5	0.001257	0.001154
rbg021.9.tw	20	0.000258	0.000151	0.000115	rbg125a.tw	126	>5	0.001978	0.002008
rbg021.tw	20	>5	>5	0.000140	rbg132.2.tw	131	>5	0.002123	0.002050
rc_201.1.txt	20	3.243302	0.000139	0.000118	rbg132.tw	131	>5	0.002077	0.002067
n20w100.003.txt	21	0.492482	>5	0.342935	rbg152.3.tw	151	>5	0.002620	0.002673
n20w20.001.txt	21	1.045714	0.048693	0.275949	rbg152.tw	151	>5	0.002690	0.002655
n20w20.002.txt	21	0.006187	4.025388	0.050206	rbg172a.tw	173	>5	0.003695	0.003601
n20w20.003.txt	21	0.310982	2.607636	0.000149	rbg193.2.tw	192	>5	0.004271	0.004554
n20w20.004.txt	21	>5	0.000150	0.000126	rbg193.tw	192	>5	0.004280	0.004305
n20w20.005.txt	21	>5	>5	0.000152	rbg201a.tw	202	>5	0.004749	0.004726
n20w40.002.txt	21	0.224788	>5	0.284337	rbg233.2.tw	232	>5	0.006653	0.006219
n20w60.004.txt	21	0.000452	>5	0.004055	rbg233.tw	232	>5	0.006314	0.006395
n20w60.005.txt	21	>5	>5	0.000156					
Среднее время работы (с превышением)							3.549345	1.728938	0.015658
Среднее время работы (без превышения)							0.384279	0.229702	0.015658

Таблица 1: Времена работы алгоритма с различными метриками близости

6.2 Муравьиный алгоритм

Проведено исследование скорости работы алгоритма при различных входных данных, то есть рассматривается зависимость времени работы алгоритма при определенных параметрах от количества городов.

Параметры алгоритма были выбраны так: $n = 2000$ - количество итераций(поколений); $m = 20$ - количество муравьев;
$\gamma = 0.6, \beta = 0.4$ - коэффициенты эвристик;
$\sigma = \alpha = 0.01$ - коэффициенты кривизны функции;
$p = w = 0.1$ - коэффициенты испарения феромона;
$q_0 = 0.7$

filename	cities	time	filename	cities	time
rc_206.1.txt	4	1.233429193496704	n20w80.001.txt	21	15.099793195724487
rc_207.4.txt	6	2.7110650539398193	n20w80.002.txt	21	15.575810432434082
rbg010a.tw	11	6.663529396057129	rbg020a.tw	21	18.0367271900177
rc_202.2.txt	14	10.536027669906616	rc_204.3.txt	24	29.11821413040161
rc_205.1.txt	14	9.446940422058105	rbg027a.tw	28	28.922384023666382
rc_203.4.txt	15	12.011343479156494	rbg031a.tw	32	37.45668601989746
rbg017.2.tw	16	15.189351558685303	rbg033a.tw	34	41.64402890205383
rbg017.tw	16	13.502728700637817	rbg034a.tw	35	42.67793893814087
rbg016a.tw	17	13.041439294815063	rbg035a.2.tw	36	49.59514665603638
rbg016b.tw	17	13.097707033157349	rbg035a.tw	36	45.24809908866882
rbg017a.tw	18	14.811495542526245	rbg038a.tw	39	50.876599073410034
rc_203.1.txt	19	17.431480169296265	n40w20.003.txt	41	41.54260563850403
rbg019a.tw	20	17.026967525482178	rbg040a.tw	41	57.52219533920288
rbg019b.tw	20	16.861271381378174	rbg041a.tw	42	58.508734703063965
rbg019c.tw	20	16.947712898254395	rbg042a.tw	43	59.65052556991577
rbg019d.tw	20	18.060019731521606	rbg048a.tw	49	66.41468691825867
rbg021.2.tw	20	19.056491374969482	rbg049a.tw	50	68.32420229911804
rbg021.3.tw	20	19.344409942626953	rbg050a.tw	51	86.53223896026611
rbg021.4.tw	20	20.452852964401245	rbg050b.tw	51	68.18653345108032
rbg021.5.tw	20	21.338858366012573	rbg050c.tw	51	73.46965003013611
rbg021.6.tw	20	22.007160663604736	rbg055a.tw	56	95.98422694206238
rbg021.7.tw	20	23.427610874176025	rbg067a.tw	68	131.5631387233734
rbg021.8.tw	20	23.9029700756073	rbg092a.tw	93	205.7767629623413
rbg021.9.tw	20	23.741316318511963	rbg125a.tw	126	404.130952835083
rbg021.tw	20	16.854703426361084	rbg132.2.tw	131	463.7632427215576
rc_201.1.txt	20	15.413067817687988	rbg132.tw	131	428.71857213974
n20w100.003.txt	21	16.3936288356781	rbg152.3.tw	151	589.1030662059784
n20w20.001.txt	21	15.540846109390259	rbg152.tw	151	546.417882680893
n20w20.002.txt	21	15.524999856948853	rbg172a.tw	173	696.5723135471344
n20w20.003.txt	21	15.80284070968628	rbg193.2.tw	192	931.846268415451
n20w20.004.txt	21	15.596222877502441	rbg193.tw	192	868.2912669181824
n20w20.005.txt	21	15.443735837936401	rbg201a.tw	202	921.7013037204742
n20w40.002.txt	21	15.307519435882568	rbg233.2.tw	232	1320.7432656288147
n20w60.004.txt	21	16.74561333656311	rbg233.tw	232	1356.1015090942383
n20w60.005.txt	21	16.05206036567688			

Таблица 2: Времена работы муравьиного алгоритма

Как и ожидалось, с ростом количества городов увеличивается и время работы алгоритма, это связано с тем, что муравью приходится просчитывать вероятности перехода к большему числу городов на

каждом шаге. Чтобы снизить время работы алгоритма, можно понизить количество муравьев или уменьшить количество итераций. Этот вопрос мы сейчас и рассмотрим.
Посмотрим, как на решение влияет увеличение количества муравьев. Для этого построим график зависимости ошибки полученного решения от количества муравьев.
Ошибку будем считать так:

$$Error = \frac{R^* - R}{R^*}$$

где R^* - значение функции цели при оптимальном решении,
R - значение функции цели при полученном решении

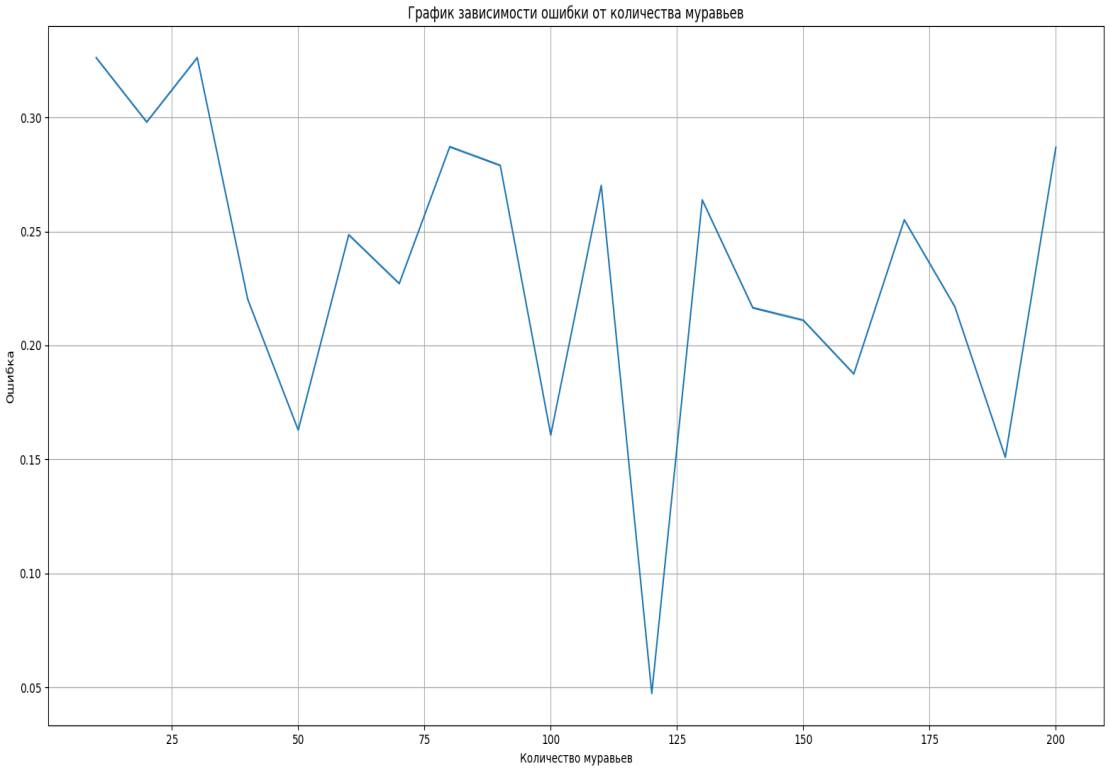


Рис. 1: График зависимости ошибки от количества муравьев

По графику сложно говорить о какой-то постоянной зависимости, т.е. увеличение количества муравьев не гарантирует улучшения работы алгоритма, но мы точно можем сказать, что чем больше муравьев, тем дольше будет работать алгоритм. Поэтому важно обращать внимание на данный параметр алгоритма.

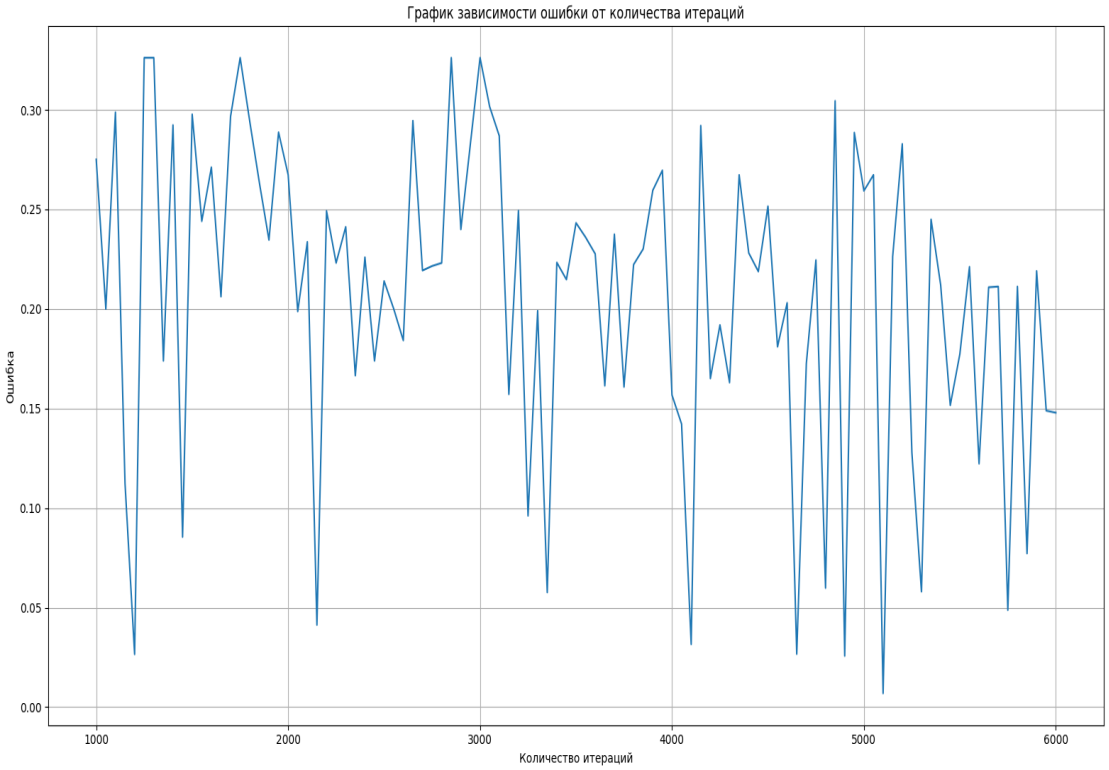


Рис. 2: График зависимости ошибки от количества итераций(поколений)

Из графика видно, что чем больше поколений, тем меньше ошибка, хотя всегда возможны "выбросы когда при маленьком числе поколений результат может быть лучше, чем при большом.

7 Выводы

В ходе нашей работы были изучены следующие алгоритмы:

ближайший сосед с возвратами

муравьиная колония

Алгоритм ближайшего с возвратами является модификацией полного перебора со следующей эвристикой: при неуспешном нахождении пути происходит откат в предыдущий город. Если учитывать текущую стоимость поездки и сравнивать ее с наилучшей стоимостью полного пути на данный момент, получится метод ветвей и границ [6].
Может показаться, что данный алгоритм дает результат лучше, чем алгоритм, который выбирает следующий город случайно, однако это не так: поскольку выбираются ближайшие соседи, наикратчайшие пути разбираются вначале, оставляя в конце одни из самых длинных [4]. Недостаток алгоритма состоит в том, что худший случай вырождается в полный перебор [3].

Проведенные эксперименты показывают, что муравьиные алгоритмы находят хорошие маршруты коммивояжера за адекватное время, однако эти маршруты не обязательно оптимальные, лишь приближенные. Также эффективность муравьиных алгоритмов увеличивается с ростом размерности задачи оптимизации [7].

Выделим ещё некоторые достоинства: [8]

+ Возможность вести параллельные расчеты;
+ Возможность работать с динамическими данными(т.е. адаптация под динамические условия задачи, например, изменяемые расстояния между городами);

Также стоит отметить и недостатки:

- Получение ряда случайных решений;
- Изменение распределения вероятностей с каждой новой итерацией алгоритма;

8 Реализация

Курсовая работа выполнена на языке программирования Python(3.7) с использованием следующих библиотек: Numpy, Matplotlib.
Отчет написан в онлайн редакторе LaTeX - Overleaf.
Ссылка на репозиторий с исходным кодом:
https://github.com/PopovIV/TSPTW

Используемая литература

[1] The Traveling Salesperson Problem, Time Complexity
https://nbviewer.org/url/norvig.com/ipython/TSP.ipynb#Complexity-of-alltours_tsp (дата обращения 20.11.2021)

[2] Backtracking Correctness
https://jeffe.cs.illinois.edu/teaching/algorithms/book/02-backtracking.pdf (дата обращения 20.11.2021)

[3] How to calculate time complexity of backtracking algorithm?
https://stackoverflow.com/a/20050433 (дата обращения 20.11.2021)

[4] The Greedy Algorithm for the Symmetric TSP
https://www.cs.rhul.ac.uk/~gutin/paperstsp/stspgreedy2.pdf (дата обращения 20.11.2021)

[5] Backtracking
https://www.win.tue.nl/~kbuchin/teaching/2IL15/backtracking.pdf (дата обращения 20.11.2021)

[6] An Exact Algorithm for the Time-Constrained Traveling Salesman Problem
https://pubsonline.informs.org/doi/pdf/10.1287/opre.31.5.938 (дата обращения 21.11.2021)

[7] Муравьиные алгоритмы
https://www.researchgate.net/publication/279535061_Stovba_SD_Muravinye_algoritmy_Exponenta_Pro_Matematika_v_prilozeniah_-_2003_-_No4_-_S_70-75(дата обращения 25.11.2021)

[8] Об алгоритмах решения задачи коммивояжера с временными ограничениями https://www.ivtio.sfedu.ru/lib/15/1-1(15)2014.pdf(дата обращения 29.11.2021)