УДК 519.168
ББК 22.151.5

С. А. Лычко

Использование метода имитации отжига для решения задачи коммивояжёра

Московский политехнический университет
г. Москва

***Аннотация**: В работе рассмотрен способ применения метода имитации отжига для решения задачи коммивояжёра. Рассмотрены его преимущества, были сделаны выводы по применимости данного алгоритма.*

***Ключевые слова**: имитационное моделирование, методы оптимизации, алгоритм имитации отжига, задача коммивояжёра, поиск пути*

Введение

Алгоритм имитации отжига — широко применяемый метод имитационного моделирования. Данный алгоритм основывается на имитации физического процесса, происходящего при кристаллизации металлов.

Предполагается, что: атомы уже выстроились в кристаллическую решётку, но ещё допустимы переходы отдельных атомов из одной ячейки в другую; что процесс протекает при постепенно понижающейся температуре. Переход атома из одной ячейки в другую происходит с некоторой вероятностью, причём вероятность уменьшается с понижением температуры. Устойчивая кристаллическая решётка соответствует минимуму энергии атомов, поэтому атом либо переходит в состояние с меньшим уровнем энергии, либо остаётся на месте [2].

Фактически, данный метод представляет собой метод многопараметрической оптимизации и является одним из примеров метода Монте-Карло. Алгоритм имитации отжига схож с алгоритмом градиентного спуска, однако сходится в локальных экстремумах значительно реже за счёт использования вероятностных составляющих.

Краткое описание алгоритма: для работы алгоритма требуется 3 функции, которые указаны на рисунке 15:

Рисунок 15 — Функции алгоритма имитации отжига.

Функция энергии (оптимизируемой величины) E, функция изменения температуры во времени T (данное функция монотонно убывает) и функция смены состояния F.

3. Ввод минимальной температуры t_{min} и начальной t_1

4. $t_i = t_{max}$

5. Пока $t_i > t_{min}$

   5.1. $s_c = F(s_{i-1})$

   5.2. $dE = E(s_i) - E(s_{i-1})$

   5.3. Если $dE < 0$, то $s_i = s_c$

   5.4. Иначе $s_i = s_c$ с вероятностью $P = e^{-dE/t}$

   5.5. $t_{i+1} = T(i)$

6. Вернуть si

Блок-схема работы данного алгоритма представлена на рисунке 16.

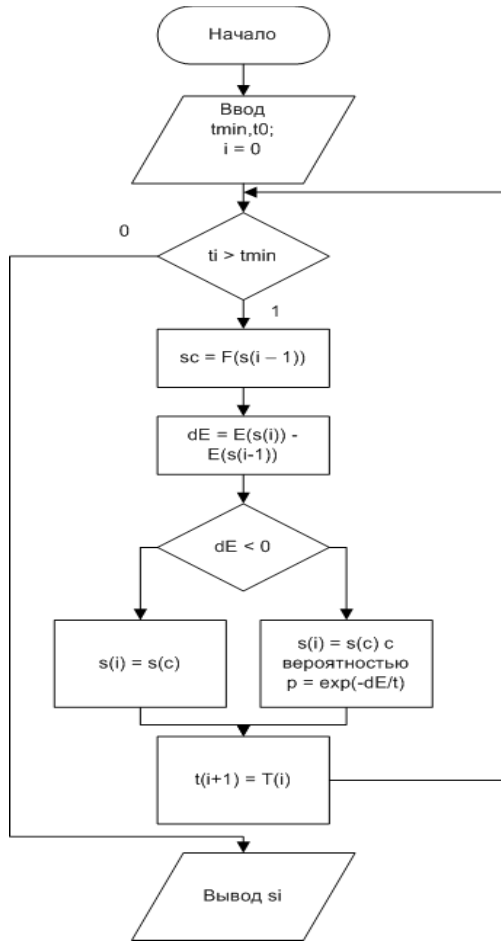

Рисунок 16 — Блок-схема работы алгоритма имитации отжига.

Задача коммивояжёра

Задача коммивояжёра — классическая задача комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город.
В условиях задачи указываются критерий выгодности маршрута (кратчайший,

самый дешёвый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. Можно рассмотреть задачу на следующем примере: «Представьте себе, что Вы — странствующий торговец и хотите предложить свой товар жителям каждого города в стране. Путешествия отнимают много сил и времени, поэтому логично, что Вы хотите составить свой маршрут таким образом, чтобы расстояние, которое предстоит преодолеть, было минимальным. Этот маршрут и предстоит отыскать».

Необходимо найти кратчайший путь, проходящий через каждый город и заканчивающийся в точке отправления. В такой постановке задача называется замкнутой задачей коммивояжёра.

Особенностью данной задачи является отсутствие эффективного алгоритма её решения, а также невозможность её решения перебором за обозримое время при достаточно большом количестве городов [2]. Пример случайного, неоптимального пути показан на рисунке 17; на рисунке 18 представлен путь после оптимизации.

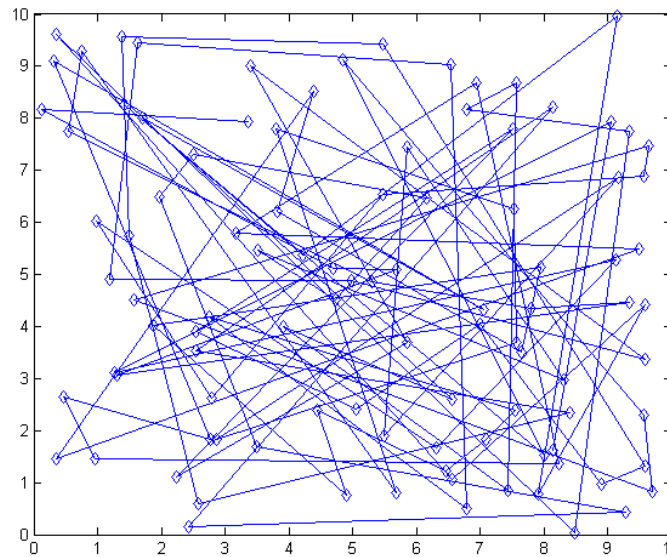

Рисунок 17 — Случайный путь.

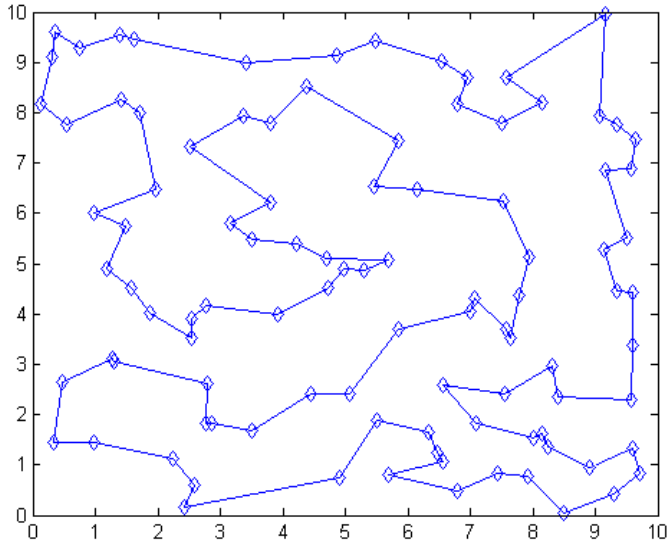

Рисунок 18 — Оптимизированный путь.

Применение имитации отжига для решения задачи коммивояжёра

Алгоритм имитации отжига удобно применять для решения задачи коммивояжёра [3]. Представим путь как последовательность пар координат всех городов. Определим начальное состояние как случайный возможный цикл, проходящий через все города. Так как мы оптимизируем (минимизируем) длину пути, то значение энергии будет представлять собой сумму евклидовых расстояний между городами.

Функция изменения температуры может иметь нелинейный или линейных характер, но в любом случае она должна убывать с каждой итерацией, иначе алгоритм может зациклиться. Один из вариантов данной функции возвращает значение, прямо пропорциональное начальной температуре и обратно пропорциональное номеру итерации. Можно усложнить эту функцию, сделав уменьшение температуры пропорциональным уменьшению энергии.

Выбор функции смены состояния может очень сильно повлиять на качество работы алгоритма. Простейшим вариантом является смена двух точек пути местами, однако есть более эффективные подходы. Одна из форм функции смены состояния, иногда называемая "зеркалом", реализуется следующим образом: выбираются 2 случайные точки пути, участок пути между ними инвертируется.

Также алгоритм требует правильного выбора начальной и минимальной температуры. Следует принимать во внимание, что чем больше температура, тем больше шанс перехода в состояние с большей энергией, что соответствует ухудшающему результат шагу, который, однако, может вывести алгоритм из локального экстремума. Кроме того, минимальная температура не должно быть отрицательной и должна быть достижима с помощью выбранной функции уменьшения температуры.

Следует понимать, что алгоритм имитации отжига в большинстве случаев будут давать не оптимальный путь, а достаточно оптимальный, то есть алгоритм не будет сходиться в точке с наименьшей энергией, но значение энергии будет близко к наименьшей.

Программная реализация

Для программной реализации алгоритма имитации отжига в рамках решения алгоритма коммивояжёра был выбран язык программирования Python версии 3.7. Данный язык удобен для быстрого прототипирования приложений из-за высокой скорости разработки, однако скорость интерпретации приложений невысока, но в демонстрационной программе она не столь важна. Для визуализации хода работы алгоритма используется библиотека Matplotlib [4].

Листинг итогового кода программы:

Листинг 1 — Основной код реализации алгоритма

```
# -*- coding: utf-8 -*-
from random import randint, shuffle, random
from math import sqrt, exp
from matplotlib.path import Path
```

```
from matplotlib import pyplot as pp, patches as
patches


# размер "территории"
_dims = (15, 15)
# количество городов
_size = 150
# начальное и минимальное значение температуры
_t0, _tmin = 50, 0.0005
 inf = 2**63


# ------------------------------------------------------


def getCities(dims, size):
    """
    Функция принимает размеры поля, необходимое коли-
чество городов,
    и возвращает случайный список городов в виде
списка пар координат
    """
    cities = []
    xmx, ymx = dims
    while len(cities) < size:
        city = (randint(0, xmx), randint(0, ymx))
        # проверка на уникальность города
        if city not in cities:
```

```
            cities.append(city)
    return cities


def getRndPath(cities):
    """
    Функция получает начальное состояние алгоритма
в виде случайного пути через все города
    """
    path = []
    while len(cities) > 0:
        path.append(cities.pop())
        shuffle(cities)
    return path


def visualise(path, save = False, name = ©test©):
    """
    Визуализация состояния
    """
    path = Path(path + [path[0]])


    fig = pp.figure()
    ax = fig.add_subplot(111)
    patch = patches.PathPatch(path, fill = False)
    ax.add_patch(patch)
    ax.set_xlim(-1,_dims[0] + 1)
```

```
    ax.set_ylim(-1, _dims[1] + 1)


    if save:
        pp.savefig(f©{name}.png©, dpi = 200)


    pp.show()
# ------------------------------------------------------


# функция энергии
def E(path):
    """
    Функция энергии
    Функция принимает путь в виде списка координат го-
родов и возвращает
    длину пути.
    """
    path_len = 0
    for i in range(1, len(path)):
        xcurr, ycurr = path[i]
        xrec, yrec = path[i-1]
        # к длине пути добавляется расстояние между
текущим городом и предшествующим
        path_len += sqrt((xcurr Ð xrec)**2 + (ycurr Ð
yrec)**2)
    # к длине пути добавляется длина между первым
и последним городом, кольцо замыкается
```

```
    x0, y0 = path[0]
    xlast, ylast = path[-1]
    path_len += sqrt((x0 Ð xlast)**2 + (y0 Ð
ylast)**2)
    return path_len


def T(recent_temp, t0, iterN):
    """
    Функция уменьшения температуры.
    Принимает предыдущее  и начальное значение темпе-
ратуры и возвращает новое.
    """
    return t0/iterN


def S(path):
    """
    Функция смены состояния
    Принимает текущий путь, возвращает возможный путь
    """
    sc = path.copy()
    fst, snd = randint(0, len(path) Ð 1), randint(0,
len(path) Ð 1)
    sc[fst], sc[snd] = sc[snd], sc[fst]
    return sc
```

```
#Основной код алгоритма
# ================================================

#подготовка начального состояния
cities = getCities(_dims, _size)
s = getRndPath(cities)
visualise(s)
# счётчик числа итераций
iterN = 1

t = _t0
Elast = inf

#цикл работы алгоритма
while t > _tmin:
    Elast = E(s)
    sc = S(s)
    dE = E(sc) Ð Elast
    if dE <= 0:
        s = sc
    else:
        P = exp(-dE/t)
        if random() < P:
            s = sc
```

```
    t = T(t, _t0, iterN)
    iterN += 1


    if iterN % 100 == 0:
        print(Elast)
# ================================================
```

Тестирование работы алгоритма

Тестирование алгоритма на графе с 20 вершинами (городами), показало неплохой результат. Его результат представлен на рисунке 19; затраченное на работу время составило 7,7 условных единиц. Но при использовании алгоритма со 150 точками, результат, отражённый на рисунке 20, получается посредственным. При этом время работы алгоритма значительно увеличилось и составило 47,9 условных единиц.

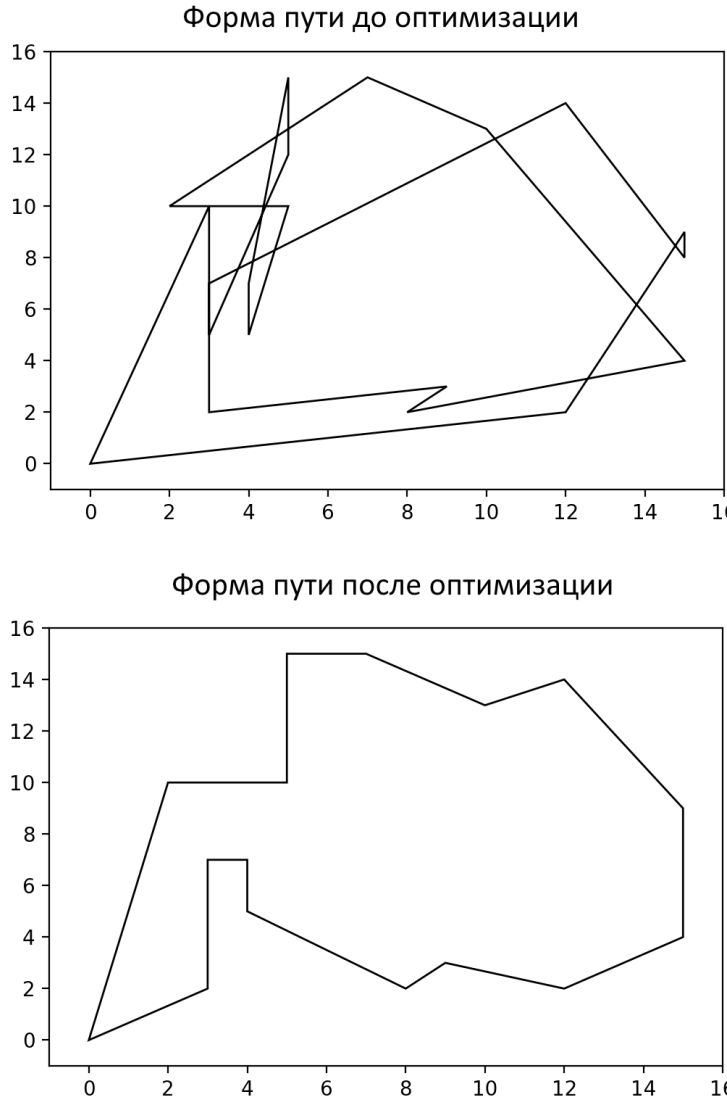

Рисунок 19 — Результаты применения алгоритма при 20 городах.

Рисунок 20 — Результаты применения алгоритма при 150 городах.

Изменив функцию смены состояния следующим образом можно улучшить результат:

Листинг 2 — Основной код реализации алгоритма

```
def S(path):
    """
    Функция смены состояния
    Принимает текущий путь, возвращает возможный путь
    """
    sc = path.copy()
    fst, snd = randint(0, len(path) Ð 1), randint(0,
len(path) Ð 1)
    sc[fst:snd] = reversed(sc[fst:snd])
    return sc
```

Полученный за время 46.54 условных единиц результат, показанный на рисунке 21, является полностью удовлетворительным.

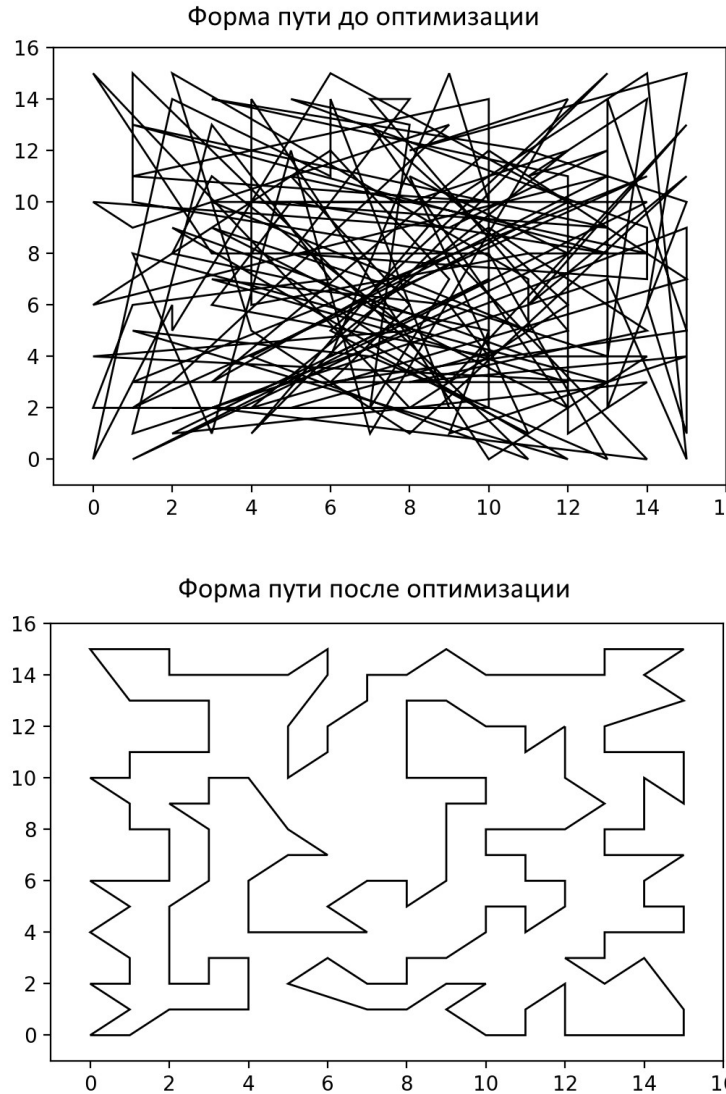

Рисунок 21 — Результаты применения алгоритма при 150 городах с обновлённой функцией смены состояния.

Выводы

Был рассмотрен метод имитации отжига для решения задачи коммивояжёра. Опробованы разные параметры алгоритма; с помощью алгоритма имитации отжига был получен удовлетворительный результат. Кроме того, алгоритм справился с задачей, которую невозможно решить методом грубой силы за обозримое время (при 150 городах).

Литература

1. Harbrahabr. Введение в оптимизацию. Имитация отжига [Электронный ресурс]. — URL: https://habr.com/ru/post/209610/ (дата обращения: 30.05.2019).
2. Ананий В. Левитин. Алгоритмы: введение в разработку и анализ. / Ананий В. Левитин, Мария Левитина. — Вильямс, 2006. — 576 с.
3. Сравнительный анализ методов решения задачи коммивояжера для выбора маршрута прокладки кабеля сети кольцевой архитектуры // Молодежный Научно-Технический Вестник. — 2013. — № 11. — С. 32.
4. Overview — Matplotlib 3.1.0 documentation [Электронный ресурс]. — URL: https://matplotlib.org/3.1.0/contents.html (дата обращения: 30.05.2019).