Problem

Travelling salesman. Единственная математическая задача) Вам дана карта, на которой отмечено N поселений. Поселения находятся на различном расстоянии друг от друга. Торговец отправляется из поселения N0. Предложите алгоритм, который позволит найти оптимальный маршрут для обхода всех N поселений и вернуться в точку старта.

Ответ должен содержать один или несколько алгоритмов (можно псевдокод) и пояснение о эффективности данных решений.

Solution

Если $D_{ji} = D_{ij}$ и расстояния между любыми городами i, j, k удовлетворяют неравенству треугольника: $D_{ij} + D_{jk} D_{ik}$, то задачу коммивояжёра называют метрической.
Вот несколько возможных алгоритмов:

I) Жадный алгоритм Nearest Neighbour (NN). На каждой итерации жадного алгоритма выбираем ближайшую вершину к последней выбранной и добавляет её в путь.
Оценка эффективности: полученный путь не может быть более, чем вдвое хуже оптимального возможного.

II) Алгоритм состоит из нескольких пунктов:
а) Ищем минимальное остовное дерево (например алгоритмом Краскала);
б) Выбираем начальную вершину, проводим от неё поиск в глубину;
в) Убираем из полученной последовательности все повторяющиеся
вершины

Этот алгоритм 2-приближенный?
Стоимость вывода, полученного с помощью вышеуказанного алгоритма, никогда не более чем вдвое превышает стоимость наилучшего возможного выхода. Давайте посмотрим, как это гарантируется приведенным выше алгоритмом.
– Кратчайший гамильтонов цикл не меньше MST.
– Длина эйлерова маршрута в G равна двум MST.
– Неравенство треугольника полученный гамильтонов цикл не длиннее эйлерова обхода не длиннее двух MST
Из вышеприведенных утверждений мы можем сделать вывод, что стоимость , полученная с помощью приближенного алгоритма, никогда не более чем вдвое превышает стоимость наилучшего возможного решения.

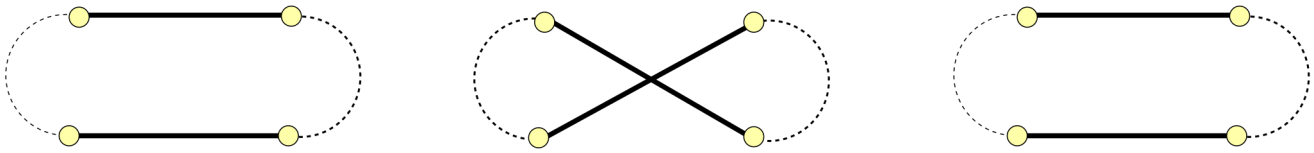
III) Алгоритм Кристофидеса. Схож с прошлым пунктом, но после нахождения остовного дерева

Definition 1. *Паросочетание – подмножество ребер графа, такое, что никакие два ребра не инцидетны какой-либо одной вершине.*

Definition 2. *Совершенное паросочетание – паросочетание, покрывающее все вершины графа*

б) Для всех вершин нечётной степени строится идеальное паросочетание, после чего ребра из него добавляются в дерево
в) В полученном графе ищется эйлеров цикл (теперь он точно есть, так как все вершины имеют чётную степень), который и является ответом. Оценка в данном случае: полученный путь не может быть более, чем в полтора раза хуже оптимального возможного. В общем случае из оценки неизвестно, какой из алгоритмов найдёт лучший ответ, поэтому стоит запустить каждый и найти лучший.

Remark 3. *Несложно видеть, что в евклидовой задача коммивояжёра (ETSP) оптимальный путь не должен иметь пересечений. Действительно, пусть есть некоторый путь, изображённый ниже на первом рисунке. Пунктиром обозначены остальные его участки (т.е. приведены только 4-е города). Если перебросить два участка пути (как на второй картинке), то общий путь вырастет (сумма диагоналей 4-угольника всегда длиннее суммы двух противоположных сторон):*



Эвристические методы

Эвристические методы, обычно, существенно быстрее точных, однако они не гарантируют оптимальности найденного решения. Результат их комбинации может далее использоваться как первое приближение для последующего улучшения, например, при помощи поиска с возвратом:
Жадный алгоритм при выборе очередного города берёт ближайший не посещённый до этого город.
Метод шнурка - геометрическая вариация жадного алгоритма, в которой города охватываются замкнутым контуром. Он постепенно растягивается, стараясь пройти через все города, минимальным образов увеличив свою длину.
Скользящий перебор переставляет местами города из небольшой части пути. Затем такое "окно перебора" скользит вдоль всего пути. Метод имеет различные вариации и оказывается эффективным способом улучшения решения, найденного предыдущими двумя эвристическими методами.

Случайное блуждание (Random Walk) – Выберем случайный путь и посчитаем его длину.
– Сделаем случайное локальное изменение и пересчитаем длину.
– Повторим предыдущий шаг много раз и выберем наилучший ответ.

```
s := randomState
f := score (s)
fBest := f; sBest := s
for step = 0; step < steps; step++:
    c := randomLocalChange
```

```
    f := applyAndRescore (s, c)
if fBest > f:
    fBest := f; sBest := s
```