Генетический алгоритм для решения задачи коммивояжера

Описание

В нашей постановке задачи коммивояжера граф задается множеством вершин на плоскости, ребрами могут являться прямые, соединяющие любые две вершины в графе.

В ходе работы предлагается реализовать:

- Вычисление фитнесс-функции (длины пути)
- Представление индивида (кодирование пути)
- Генерацию индивида
- Оператор мутации
- Оператор кроссовера

Также необходимо подобрать параметры генетического алгоритма для лучшей сходимости.

Реализация

TspSolution

В качестве реализации внутреннего представления решения был ииспользован класс-обертка над массивом чисел, размером соответствующего количеству вершин в графе и хранящему номера вершин в порядке обхода. Инициализация индивида происходит либо случайным перемешиванием вершин, либо по заданному массиву, хранящему порядок.

TspFitnessFunction

При инициализации fitness function создается объект класса Task, хранящий в себе множество вершин графа с координатами.

При вычислении fitness function происходит обход графа по порядку заданному `TspSolution` и суммирование всех расстояний.

TspFactory

Фабрика индивидов - пустышка, вызывающая конструктор для случайной генерации индивида.

TspMutation

`TspMutation` задается двумя генераторами целых чисел:

- Генератор количества мутаций
- Генератор расстояний между переставляемыми позициями

При мутации одного объекта мы сэмплируем количество перестановок, а затем нужное количество раз повторяем выбор двух элементов массива и меняем их местами.

TspCrossover

В качестве кроссовера было использовано сохранение середины. Выбирая левую и правую границу сохранной зоны, мы переносим ее из родителя в ребенка, а затем итерируемся по оставшимся элементам другого родителя и добавляем их в ребенка в том же порядке.

Границы сэмплируются равномерно.

Другие параметры

В качесве генераторов в мутации были выбраны 2 независимых сэмплера `PoissonGenerator(1.5)`. А первый индекс для замены генерировался равномерно.

В качестве стратегии отбора был использован `class RankSelection` из фреймворка `watchmaker`.

Использованный алгоритм решения - `GenerationalEvolutionEngine`.

Результаты

Проблема	**Размер**	**popsize;gens**	**Длина маршрута**	**Итераций до сходимости**	**Оптимальный маршрут**
xqf131	131	100 ; 100000	702,06	96555,35	564
xqg237	237	200 ; 100000	2231,01	99382,3	1019
pka379	379	200 ; 100000	4481,48	99374,6	1332
bcl380	380	200 ; 100000	5690,16	99496,1	1621
pbk411	411	200 ; 100000	5084,65	99711,0	1343

Вопросы

1. Проверить, является ли полученное решение глобальным оптимумом можно, если решить TSP точно, что явно нам не подходит (in EXP). TSP = NP-hard problem, по крайней мере в постановке с заданной границей длины пути. Если брать алгоритм для недетерминированной машины, то можно решить TSP(G, |solution| - e) и если такого пути нет, то мы нашли глобальный оптимум. Но все равно нет алгоритма, проверяющего минимальность пути за разумное время.
2. Допускать такие решения можно, однако, т.к. они не являются валидными решениями задачи, их придется отфильтровывать на этапе вычисления fitness. Очевидным образом появление таких решений сильно замедлит алгоритм, т.к. для каждого решения в популяции придется проводить валидацию, а при совсем плохом исходе невалидные решения могут заполнить собой почти всю популяцию и значительно уменьшить частоту генерации валидных решений.
3. При нахождении гамильтонова пути в графе из fitness function нужно убрать добавление замыкающего ребра. Генерацию и представление решений можно оставить без изменений, т.к. она все еще задает валидное решение задачи. Однако, стоит принять во внимание, что количество возможных решений задачи увеличится в |V| раз, т.к. при поиске цикла все циклические перестановки его вершин изоморфны с точки зрения fitness function, но из 1 циклического решения получается |V| решений для пути удалением ребра. А значит пространство поиска увеличится в разы и алгоритм будет работать медленнее.