Прошян Г.А. M4150 Генетические алгоритмы

Лабораторная работа №4. Генетический алгоитм для решения задачи коммивояжёра.

Цель работы

Получние навыков разработки эволюционных алгоритмов для решения комбинаторных задач на примере задачи коммивояжёра.

Оборудование и программное обеспечение

- Java JDK версии 1.8 и выше
- Watchmaker framework версии 0.7.1 (https://github.com/dwdyer/watchmaker)
- Шаблон проекта: https://gitlab.com/itmo_ec_labs/lab3

Представление решений (TspSolution)

Решение (объект класса TspSolution) имеет поле route типа ArrayList, хранящее гены (перестановку индексов городов). Также решение имеет вспомогательные методы для реализации генетических операторов. Генетические опреаторы не взаимодействуют с полем route напрямую: только с помощью этих методов. Таким образом достигается абстракиця и инкапсуляция. Если мы в целях повышения эффективности захотим хранить гены в другом виде, нам придется поменять реализацию интерфейса TspSolution, в дргуих классах код останется прежним.

Конструктор TspSolution(citiesNum) по умолчанию инициализирует поле route случайной перестановкой чисел от 0 до citiesNum

Мутации (TspMutation, TspScrambleMutation, TspInversionMutation, TspInsertMutation, TspSwapMutation)

В генетическом алгоритме (TspAlg) используются все виды мутаций сразу.

TspMutation - абстрактный родительский класс для мутаций вставкой, перемешиванием, инверсией и перестановкой.

Конструктор мутации принимает на вход число типа double от 0 до 1, представляющее вероятность того, что будет применена данная мутация

- TspScrambleMutation перемешивает гены случайно выбранного участка
- TspInversionMutation инвертирует гены случайно выбранного участка
- TspInsertMutation производит вставку гена, хранящегося по случайно выбранному индексу в позицию после дургого случайно выбранного индекса
- TspSwapMutation меняет местами гены, хранящиеся по двум случайно выбранным индексам

Кроссовер (TspCrossover)

TspCrossover реализует упорядоченный кроссовер:

- случайным образом выбираются индексы начала a и конца b фрагмента первого родителя
- фрагмент копируется в соответствующий участок дочернего решения, а также заносится в хэш таблицу
- производится циклический обход второго родителя, начиная с позиции b. Каждый раз, когда встречается ген, которого нет в хэш таблице, он добавляется в следующую позицию дочернего решения.

Результаты экспериментов

Имя проблемы	Размер	Параметры pop. и gen.	Длина маршрута	Количество итераций до сходимости	Оптимальный маршрут
xqf131	131	100, 1000000	795.42	997753	564
xqg237	237	100, 1000000	2188.59	994044	1019
pma343	343	100, 1000000	2641.91	988136	1368
pka379	379	100, 1000000	2901.32	991406	1332

Ответы на вопросы:

1. Можно ли определить, что полученное решение является глобальным оптимумом?

   В реальных задачах это невозможно. Несмотря на то, что глобальные алгоритмы с вероятностью, равной 1, сходятся к глобальному минимуму на бесконечном времени, когда время работы алгоритма конечно, и пока не были перебраны все варианты, достижение глобального минимума невозможно проверить.

2. Можно ли допускать невалидные решения (с повторением городов). Если да, то как обрабатывать тауие решения и как это повлияет на производительность алгоритма.

   Нельзя допускать невалидные решения. Такие решения очевидно бесполезны. Мы можем просто отсеивать их соответствующими проверками, но это приведет к крайней неэффективности генетического алгоритма. К тому же, если невалидные решения окажутся "сильнее" алгоритм может перестать генерировать валиндые решения.

3. Как изменится задача, если убрать условие необходимости возврата в конечную точку?

   В коде необходимо будет лишь убрать одно слогаемое в вычислении фитнес функции. Данная задача отличается от исходной, поэтому нужно будет искать новое решение.