**Секция «Информатика»**

**«Практическое применение генетических алгоритмов»**

Основной **целью работы** было создание простой во внедрении библиотеки, реализующей генетические алгоритмы.

Три основных класса – Generation (Поколение), Individual (Индивид) и Dna (ДНК). В них реализованы основные операции генетических алгоритмов (селекция, отбор, мутация), а также хранятся параметры для алгоритма (тип селекции, вероятность мутации и т.д.). Для внедрения генетических алгоритмов разработчику необходимо лишь создать переменную типа Generation, указав в качестве типа Individual. Если ни один из реализованных типов не походит к поставленной задаче, разработчик может создать свой класс, базирующийся на Individual из нашей библиотеки. Для этого класса ему необходимо определить только функцию подсчёта приспособленности и тип используемой ДНК, остальное наследуется от базового класса и не требует переопределения.

Настройки генетического алгоритма можно изменять в ходе работы программы (например, увеличивать с каждым новым поколением вероятность мутации). Такой подход к работе с библиотекой даёт разработчику простор для экспериментов и позволяет найти наиболее оптимальные настройки для алгоритма в поставленной задаче.

На данный момент в библиотеке кроме базовых классов также реализованы следующие **типы ДНК**:

* линейная действительная;
* линейная действительная с ограниченным диапазоном;
* линейная бинарная;
* древовидная ДНК выражения;

В библиотеке можно найти **индивидов** для решения таких задач, как:

* нахождение минимума / максимума функций;
* подбор функций под заданные значения переменных и заданный результат;
* подбор параметров машины для прохождения заданной трассы.

Библиотека поддерживает следующие **типы скрещивания**:

* Дискретная рекомбинация – для каждого гена потомка случайно выбирается номер особи, ген которой он унаследует.
* Промежуточная рекомбинация – применяется только к вещественным типам ДНК. Гены потомков вычисляются по следующей формуле: *c* = *a* + *а* × *b*, *с* – значение гена потомка, а и b соответственно значения генов первого и второго родителей.
* Линейная рекомбинация – отличается от промежуточной тем, что множитель a выбирается для потомка один раз.
* Многоточечный кроссинговер – выбираются *n* точек разреза и происходит обмен участками хромосом, ограниченными этими точками.
* Перетасовочный кроссинговер. В данном алгоритме особи, отобранные для кроссинговера, случайным образом обмениваются генами. Затем выбирают точку для одноточечного кроссинговера и проводят обмен частями хромосом. После скрещивания созданные потомки вновь тасуются.

Поддерживаемые **способы выбора родителей**:

* Панмиксия – выбираются две случайные особи и скрещиваются.
* Генотипный инбридинг – первый родитель выбирается случайным образом, а второй таким образом, что разность значений целевых функций выбранных особей минимальна.
* Фенотипный инбридинг – первый родитель выбирается случайным образом, а второй с минимальным Хемминговым расстоянием (для строк) или Евклидовым (для векторов).
* Генотипный аутбридинг – тоже самое, что и генотипный инбридинг, только берётся максимальное расстояние.
* Фенотипный аутбридинг – тоже самое, что и фенотипный инбридинг, только берётся максимальная разность значений целевой функции.

Поддерживается элитарный **тип отбора** в новую популяцию – в следующее поколение переходят n самых лучших особей.

Генетические алгоритмы рассмотрены на следующих **примерах**:

1. **Максимум/Минимум функции.** Пользователь задаёт какую-либо функцию, а задачей компьютера является нахождение её минимум/максимума. Стоит отметить, что для сложных функций результат в данном примере в большой степени зависит от мутаций, так как они помогают алгоритму находить минимум/максимум на всей области определения, а не останавливаться на локальных экстремумах.
2. **Подбор функции.** Известны значения функции в различных точках. Программе необходимо подобрать функцию, которая в данных точках принимает значения как модно ближе к заданным. В данном случае используется древовидное ДНК. Каждый элемент - какая-либо функция (*x*, sin *x* и т. д.), поэтому данная программа хорошо подбирает только те функции, где отсутствуют различные константы.
3. **Машинки.** Эта задача, пожалуй, самая зрелищная из всех примеров. Постоянной величиной здесь является случайно сгенерированная трасса, представленная ломаной. В качестве индивида здесь выступает машина. Машина представляет собой набор векторов из одной очки, на которые «натянут» многоугольник, называемый «корпусом». В некоторых вершинах этого многоугольника находятся колёса, которые соединены с вершиной с помощью пружины. ДНК машины хранит длину каждого из векторов, радиус каждого из колёс, его наличие либо отсутствие в конкретной вершине, а также жёсткость соединительной пружины. При этом для каждого из параметров пользователь задаёт минимальное и максимальное значение.

Использованная литература:

1. Панченко, Т.В. Генетические алгоритмы / Т.В. Панченко. Издательский дом «Астраханский университет» 2007.
2. Booker, L.B. Classifier Systems and Genetic Algorithms / L.B. Booker, D.E. Goldbergand J.H.Holland.
3. Kenneth De Jong. Genetic Algorithm Based Learning.
4. Kenneth De Jong. Genetic Algorithms Are NOT Function Optimizers.
5. Бедный, Ю. Д.. Применение генетических алгоритмов для генерации автоматов при построении модели максимального правдоподобия и в задачах управления / Ю. Д. Бедный, Магистерская диссертация, Санкт-Петербург, 2008