Решение системы линейных уравнений генетическими алгоритмами

...

Куликов Л.А.

Цели, средства

Изначально я перед собой не ставил задачу сделать очень точный алгоритм, так как генетические алгоритмы прямым образом связаны с эвристикой и случайным подбором. Это значит, что зачастую они дают приемлемое решение, но так бывает не всегда, здесь крайне важна оценка итогового решения человеком на разумность.

Мотивацией скорее для меня была сама идея. Применить алгоритм эволюции в специфичной области. Было любопытно узнать, что из этого выйдет.

В качестве языка программирования я взял C++, так как там есть удобный инструмент - вектор, который весьма упрощает многие моменты разработки. Но ввиду небольшого опыта, как оказалось, возникли некоторые проблемы с оптимизацией (при поиске более точного решения зачастую нужно больше времени на вычисления или более производительный процессор).

Основные проблемы, с которыми пришлось столкнуться или недостатки генетических алгоритмов:

Что было внедрено для “сглаживания” этих недостатков

Важно понимать, что в большинстве случаев генетические алгоритмы проигрывают научным. Однако их можно применять, когда время на вычисления в достатке и когда научные алгоритмы неэффективны (большие разреженные матрицы) или необходима лишь оценка решения.

Сходимость к локальному оптимуму я исправил при помощи постоянного поддержания разнообразия в популяции и перехода к лучшим особям на этапе кроссинговера. На этапе селекции если в популяции встречаются близкие по значениям x-ов члены (при скрещивании такие члены не дают новых решений), я подвергаю одного из них мутации с вероятностью MUT, заданной в исходном коде. Однако, как оказалось, это обстоятельство может косвенно влиять на поиск сверхточного решения, так как близкими я считаю тех членов у которых разность между x-ами меньше какой-то величины - это значит мы можем все время проскакивать очень маленькие интервалы, где, возможно, находится сверхточный результат.

ГА довольно неплохо себя показывают на дискретной области поиска. Но я все же решил решать уравнения в поле действительных чисел, а это значит, что область поиска бесконечна. Этот нюанс кардинальным образом я изменить не смог, однако, задание границ определенным образом периодически помогает получать достаточно точные решения.

Алгоритм

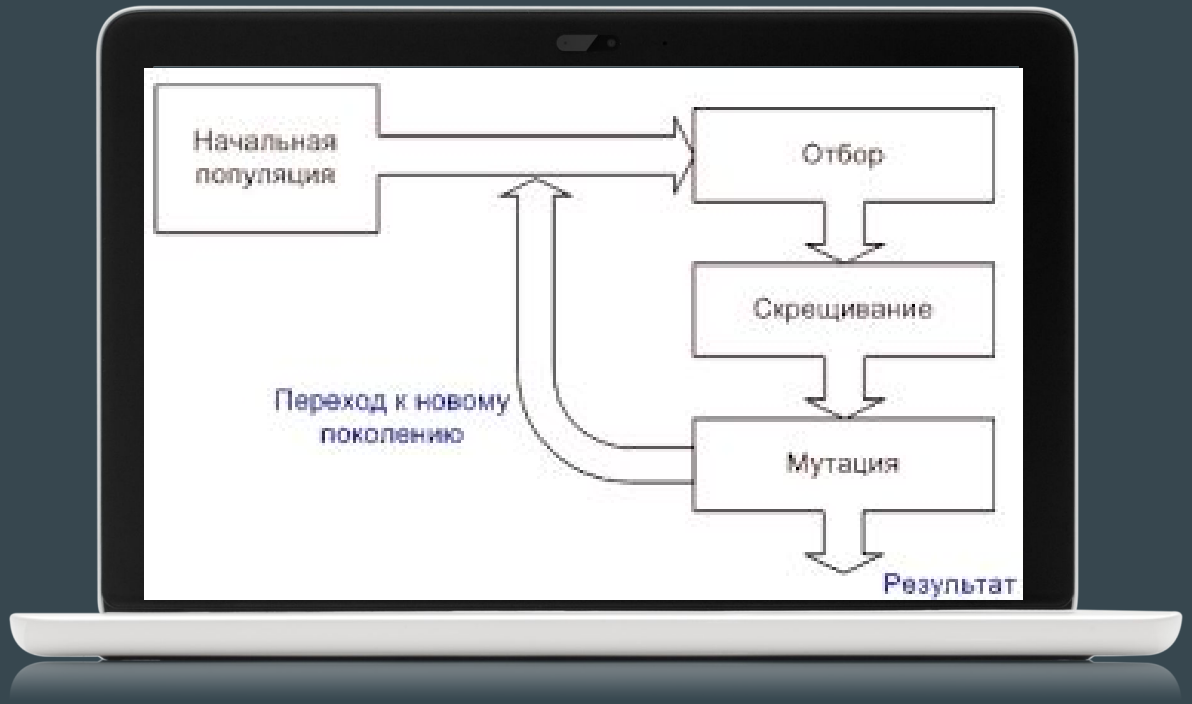
Алгоритм для решения поставленной задачи я взял из книги “Генетические алгоритмы” Курейчика В.В, В.М., и Гладкова Л.А. Алгоритм придуман Л. Дэвисом, по большому счету он так же похож на алгоритм Д. Гольдберга и Д. Холланда, так как в них есть 3 основные операции:

1. Репродукция;
2. Кроссинговер;
3. Мутация.

Репродукция - перемещение лучших особей в новое поколение.

Кроссинговер - рекомбинация “хороших” членов.

Мутация - случайное изменение характеристик заданных членов.



P.S. Подробное описание алгоритма продемонстрировано в отчете.

Каким образом я реализую репродукцию и рекомбинацию

Операцию рекомбинации в каждой итерации поколений я произвожу N/2 раз. Для этого я N раз выбираю родителей при помощи метода рулетки (описан в книге “Генетические алгоритмы”). Это значит, что каждому члену популяции я отвожу некоторый блок в массиве, размер которого пропорционален его значению функции выживания. Далее рандомом выбираю блок в этом массиве и таким образом выбираю родителей. Стоит отметить, что размножение исключительно половое, так как рекомбинация не дает новых решений при моно-скрещивании. После этого я создаю массив со всевозможными потомками данных родителей. И выбираю из этого массива только тех, чья невязка меньше, чем у обоих родителей и переношу их в следующее поколение. Если таких нет, то в дело вступает репродукция, то есть места в новом поколении занимают снова родители. Может показаться, что это хитро, но, на мой взгляд, такая селекция оптимальна, ибо считается, что жизнь со временем требует большего значения функции выживания, поэтому потомки, которые являются хуже родителей, умирают.

Пример рекомбинации:

Parent 1: 1 1 3 | 4 0 0

Parent 2: 9 9 5 | 2 3 1

Child 1: 1 1 3 2 3 1

Child 2: 9 9 5 4 0 0

Мутация

Мутация в данном алгоритме крайне важна, иначе популяцию заполнила бы моногамная система, что при любой дальнейшей рекомбинации не дало бы новых решений.

Поэтому в каждой итерации алгоритма, я произвожу проверку на хаос в популяции, таким образом близкие друг к другу элементы с определенной вероятностью становятся различными (в рамках выдачи рандомных чисел).

Однако здесь есть важный нюанс, нельзя всегда точно сказать каким должен быть коэффициент мутации. Его нужно подбирать либо из начальных условий, либо рандомно. Получая определенные решения, можно его менять и смотреть стали ли решения лучше или нет.

Спасибо за внимание!