ВВЕДЕНИЕ В ЭВОЛЮЦИОННЫЕ ВЫЧИСЛЕНИЯ

(СЕНТЯБРЬ – ДЕКАБРЬ 2008)

Акира Имада, Войцехович Леонид

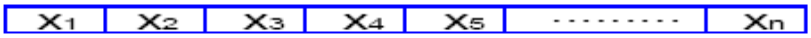
(в процессе разработки – последние изменения)

5 марта 2009 г.

1 Что представляет собой Генетический Алгоритм?

Пусть имеется набор из *n* переменных. Цель задачи – получить оптимальную последовательность из *n* переменных. Воспользуемся генетическим алгоритмом:

- Совокупность последовательностей переменных x_i образуют *популяцию*. Каждая такая последовательность называется хромосомой (или, по-другому, *особью*).



- Зададим оценку *приспособленности*, которая будет отражать “Насколько хороша каждая отдельная особь?”

Далее выполняются необходимые вычисления. Предполагается, что результат будет постоянно улучшаться при переходе от одной популяции к последующей.

1. (**Инициализация**) На этом этапе случайным образом генерируется начальная *популяция*, состоящая из *p* хромосом.

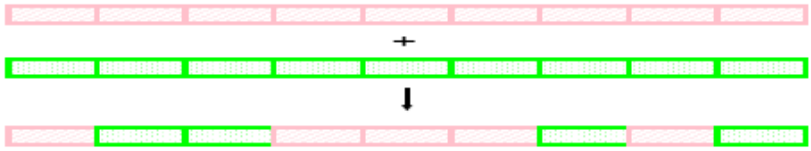
2. (**Оценка приспособленности**) Дается оценка соответствия каждой хромосомы желаемому результату и выполняется сортировка хромосом в порядке уменьшения их качественных характеристик.

3. (**Селекция**) Выбираются две хромосомы

- В нашем случае случайным образом из представителей лучшей половины популяции (Селекция Усечением – Truncate Selection).

4. (**Репродукция**) Производится потомок в результате двух следующих операций:

- Равномерное скрещивание (Uniform Crossover), например:



- Мутация

5. Формируется следующая популяция путем повтора шага 3 и 4 *n*-раз.

6. Повторяются шаги со 2-го по 5-ый до тех пор, пока не будет получено оптимальное или хотя бы максимально приближенное решение.

2 Как выбирать родительские особи?

Чем выше качественные характеристики особи, тем больше вероятность ее выбора.

Три возможных метода выполнения Селекции:

- Селекция отсечением (Truncation Selection) – самый простой метод из трех: родители выбираются только из числа наилучших особей, составляющих определенную долю популяции.

- Селекция по принципу “Рулетки” (выбор пропорционально качеству): вероятность быть выбранным пропорциональна величине оценки приспособленности.

	fitness
individual #1	0.375
individual #2	0.125
individual #3	0.125
individual #4	0.125
individual #5	0.250

Такую селекцию можно выполнить следующим образом: отсортировать имеющиеся хромосомы в порядке убывания значения их оценки приспособленности и последовательно пересчитать для каждой хромосомы значение оценки накопительным методом.

individual after sort	cumulative value of fitness
#2	0.125
#3	0.250
#4	0.375
#5	0.625
#1	1.000

Далее случайным образом присваиваем значение переменной r в диапазоне от 0 до 1. Если r<0.125, тогда выбираем образец под №1, иначе, если r<0.250, то выбираем №2, иначе, если r<0.375, то выбираем №3 и так далее…

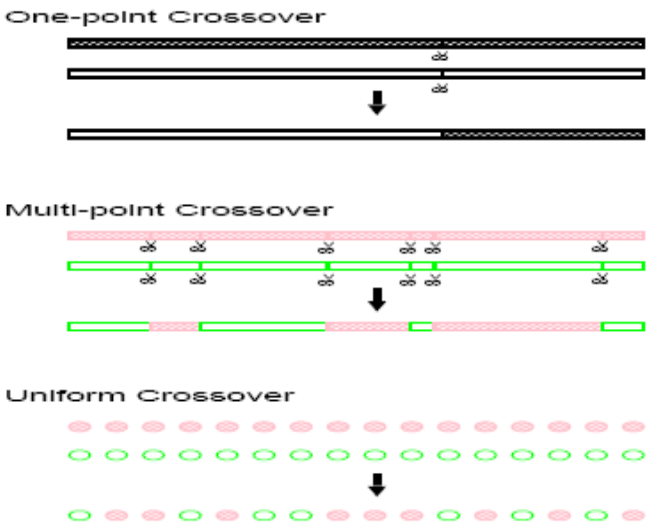
- Турнирная селекция (Tournament Selection):
  предположим, что имеется μ родительских особей, и μ потомков. Значение оценки приспособленности каждой из 2μ особей сравнивается с оценками приспособленности q особей, которые случайным образом выбираются из множества 2μ для каждой операции сравнения. Затем 2μ особей упорядочиваются согласно количеству побед, и лучшие μ образцов выживают (q-турнирная селекция).

Обратите внимание, что при селекции по принципу рулетки, существует возможность (хоть и небольшая) попадания наихудших представителей текущей популяции в следующую популяцию. А в случае селекции отсечением такое в принципе не возможно. При турнирной селекции так же существует вероятность выбора плохих детекторов, помимо самых худших q представителей популяции. Подбирая параметр q можно контролировать численность плохих детекторов в новой популяции.

3 Как из родительских особей получить потомство?

3.1 Кроссовер (Cross-over)

Так называемый кроссовер (или скрещивание) используется для генерации потомков. Рассмотрим три различных метода выполнения операции скрещивания.



3.2. Мутация (Mutation)

Мутация используется для получения новых генов. Это необходимо, чтобы избежать ситуации, когда особи популяции сходятся в *локальном минимуме* функции приспособленности. Вероятность мутации невелика – приблизительно равна 1/(количество_генов).

> *Т.е. если случайно сгенерированное в диапазоне от 0,0 до 1,0 число меньше значения вероятности мутации, то процедура мутации выполняется, в противном случае – нет.*

4 Наиболее часто используемые функции приспособленности.

Рассмотрим уже известные функции и применим генетический алгоритм.

Здесь описываются две многомерные функции, глобальные минимумы которых уже известны. Тем не менее, полезно изучить эти функции, для того чтобы понять каким образом эволюционные процессы моделируются при помощи компьютеров.

4.1 Сферическая модель (Sphere model)

Одна из простейших функций:

$$y = \sum_{i=1}^{n} x_i^2 \tag{1}$$

которая определена, к примеру, в диапазоне $x_i \in [-5.12, 5.12] \quad (i = 1, 2, \cdots, n)$. Эта функция – представление хорошо известной функции $y=x^2$ в n-мерной интерпретации. Единственный глобальный минимум находится в начале координат, а локальные минимумы отсутствуют. Поэтому полезно начать изучение генетического алгоритма именно с этой функции.

4.2 Функция Shcwefel

Функция, представленная ниже, называется Shcwefel-функцией. Она характеризуется большим количеством локальных минимумов и одним глобальным минимумом в точке начала координат. Найдите глобальный минимум, когда функция задана, допустим, в интервале $x_i \in [-500, 500]$.

$$y = \sum_{i=1}^{n} (x_i \sin(|x_i|)) \tag{2}$$

Например, для начала можно опробовать эту функцию при n=20, т.е. когда поверхность определена в 20-мерном Евклидовом пространстве. Если вы все же хотите увидеть график этой функции, то можно воспользоваться ее двумерным представлением (Рисунок 3):

$$y = x \sin(|x|) \tag{3}$$

4.3 Другие функции:

- Rastrigin функция F_6:

$$y = nA + \sum_{i=1}^{n} (x_i^2 - A\cos(2\pi x_i)), \quad x_i \in [-5.12 - 5.12].$$

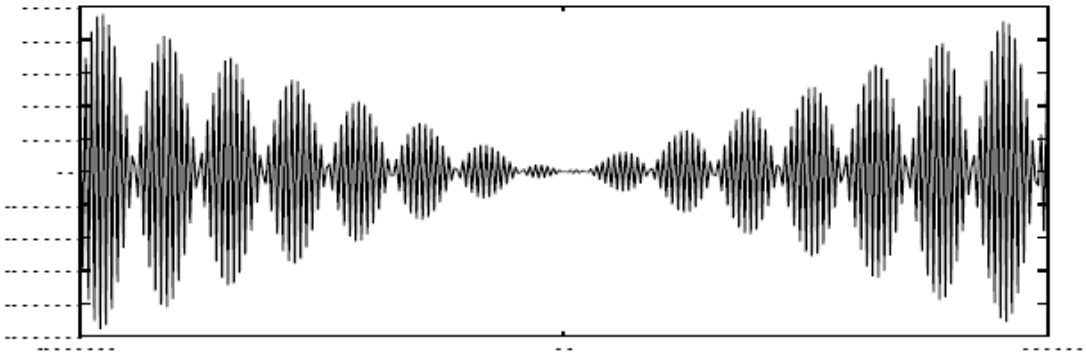


Рисунок 3: Двумерное графическое представление функции Shcwefel.

- Величина неровностей функции может настраиваться посредством параметра A.
- Пример двумерного представления (при n=1):

$$y = 3 + x^2 - 3\cos(2\pi x).$$

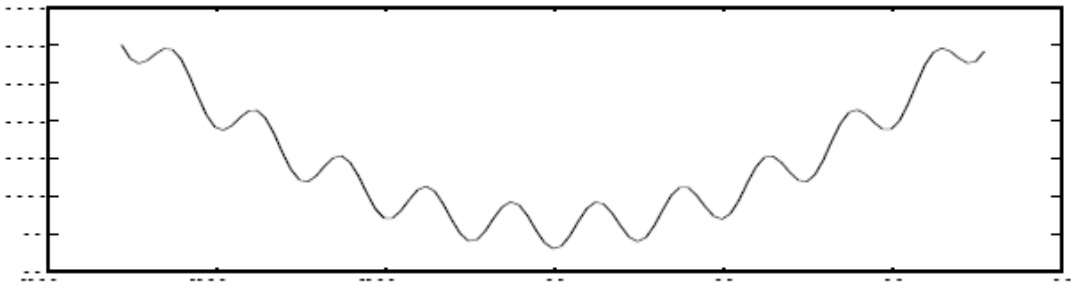


Рисунок 4: Двумерное графическое представление функции Rastrigin.

- Griewangk функция F_8:

$$y = \sum_{i=1}^{n} x_i^2/4000 - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1, \quad x_i \in [-600, 600].$$

- Двумерное представление:

$$y = x^2/4000 - \cos x + 1.$$

- Ackley функция F_9:

$$y = -20 \sum_{i=1}^{n} \exp\left(-0.2\sqrt{x_i^2/n}\right) - \exp\left(\left(\sum_{i=1}^{n} \cos 2\pi x_i\right)/n\right) + 20 + e,$$

$$x_i \in [-30, 30].$$

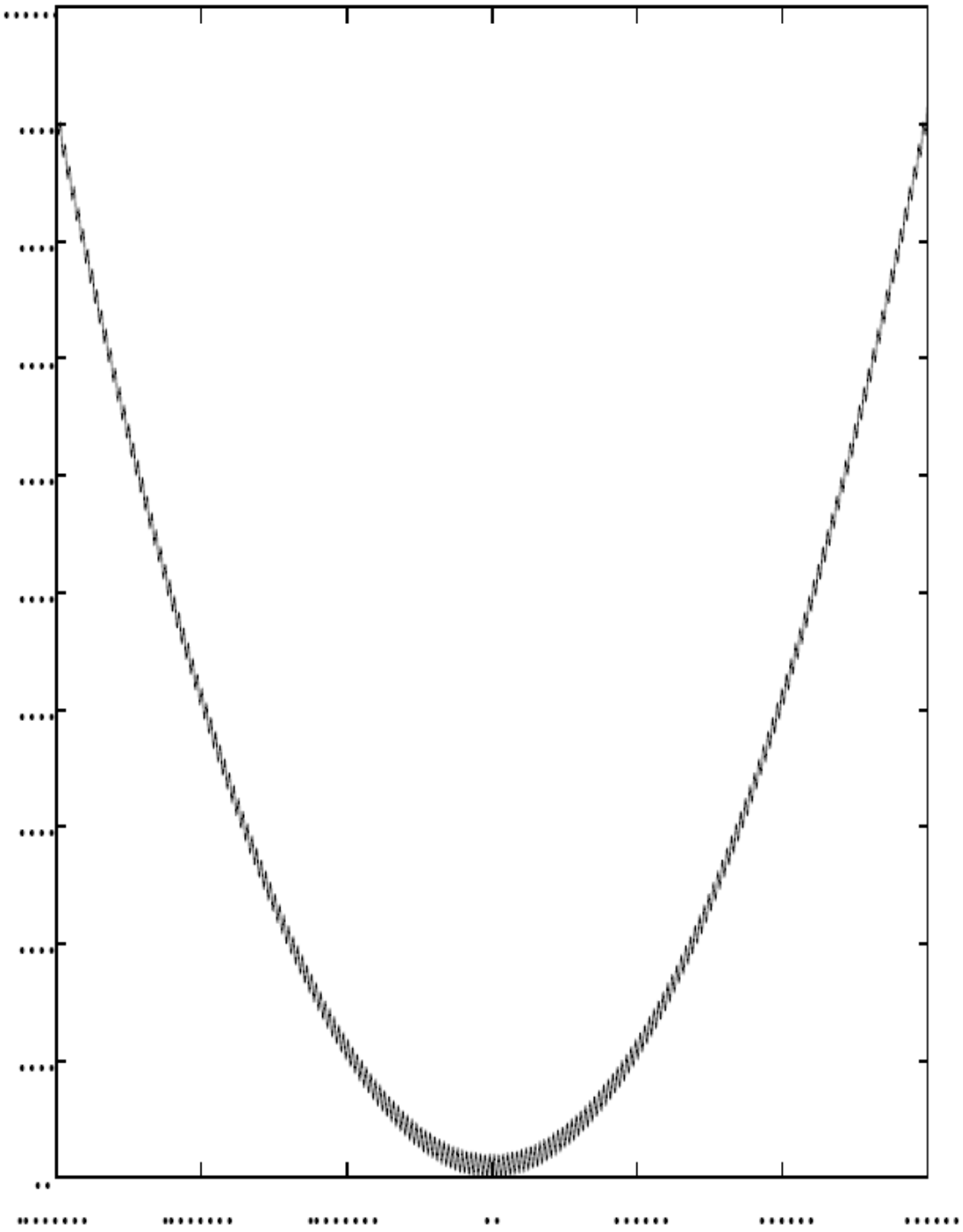


Рисунок 5: Двумерное графическое представление функции Griewangk.

- Двумерное представление:

$$y = -20\exp\left(-0.2\sqrt{x^2}\right) - \exp\left(\cos 2\pi x\right) + 20 + e.$$

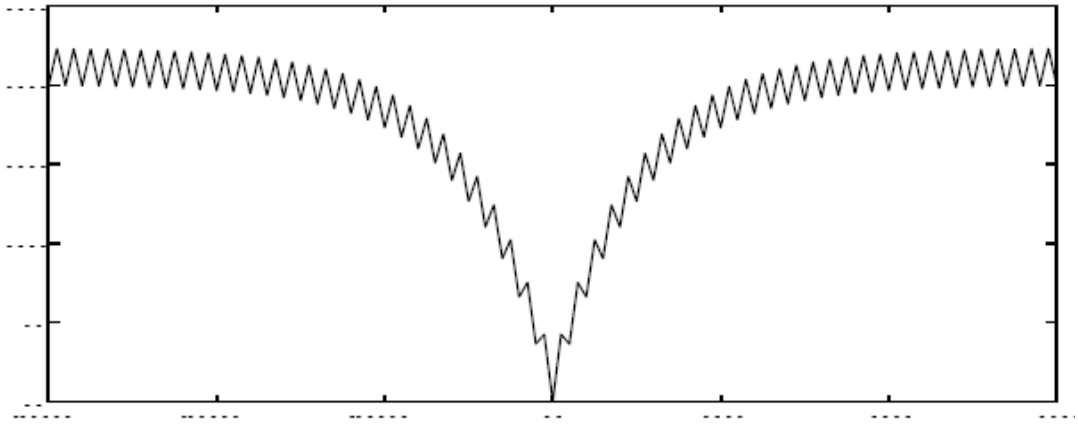


Рисунок 6: Двумерное графическое представление функции Ackley.

5 Рассмотрим простую двумерную функцию

Рассмотрим более простую функцию, определенную для одной переменной x, и изучим поведение хромосом в случае, когда функция имеет два минимума: один локальный, а второй - глобальный. Например:

$$y = x^4 - 5x^3 - 6x^2 + 8x + 15$$

где $x \in [-2, 5]$

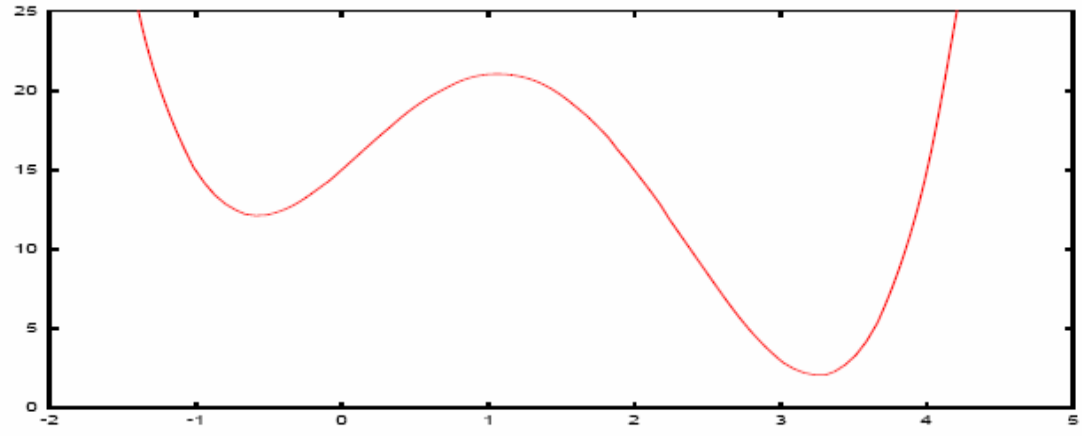


Рисунок 7: Наша следующая функция: $y = x^4 - 5x^3 - 6x^2 + 8x + 15$, где $x \in [-2, 5]$.

Попробуйте применить генетический алгоритм для поиска минимального значения y.

6 Нейронные сети для операции XOR

Это вероятно простейший пример. Рассмотрим его подробно.

Будем считать, что нейроны Маккалоха-Питта (McCulloch-Pitts) принимают значения 1 и 0. В таком случае выходное значение Y нейрона, для которого рассчитывается взвешенная сумма сигналов Xi от N входных нейронов, определяется по формуле:

$$Y = \mathrm{sgn}(\sum_{i=1}^{N} w_i X_i - \theta),$$

где sng(x)=1, если x≥0, иначе 0; wi и θ - *вес* и *порог* соответственно. Предположим, что в нашем случае нейроны принимают значения –1 и 1, а не 0 и 1. Для этого исходную формулу необходимо модифицировать следующим образом:

$$Y = 2\mathrm{sgn}(\sum_{i=1}^{N} w_i X_i - \theta) - 1.$$

6.1 Нейронная сеть для выполнения операции AND и OR

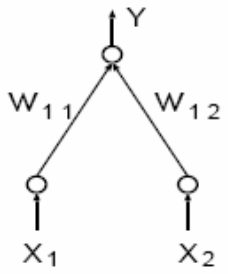
Рассмотрим нейронную сеть для моделирования логических функций AND и OR.

AND

X_1	X_2	Y
-1	-1	-1
-1	+1	-1
+1	-1	-1
+1	+1	+1

OR

X_1	X_2	Y
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	+1



Эти задачи настолько просты, что без проблем можно самостоятельно подобрать необходимые значения весовых коэффициентов. Например, w_i=0.5, w_2=0.5 и θ=0.5, для AND, и w_1=0.5, w_2=0.5 и θ=0.5 для OR.

Но почему бы ни попытаться применить эволюционный алгоритм вычислений.

6.2 Нейронная сеть для выполнения операции XOR

Попробуем применить нейронную сеть для выполнения операции XOR. В этом случае понадобится еще один слой, так называемый *скрытый слой*. Дело в том, что…

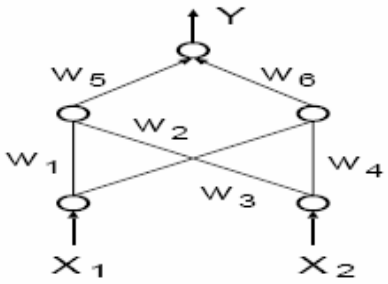
Упражнение 1. *Вычислите такие значения шести весовых коэффициентов, чтобы представленная нейронная сеть, функционировала по принципу XOR.*
Упражнение 2. *Создайте псевдокод эволюционного алгоритма для расчета этих значений весовых коэффициентов.*

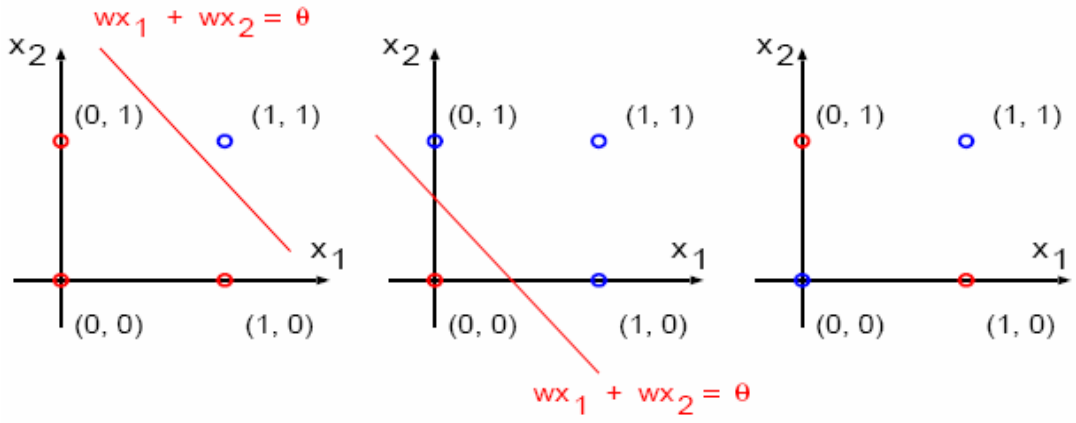
6.3 Большие и сложные нейронные сети

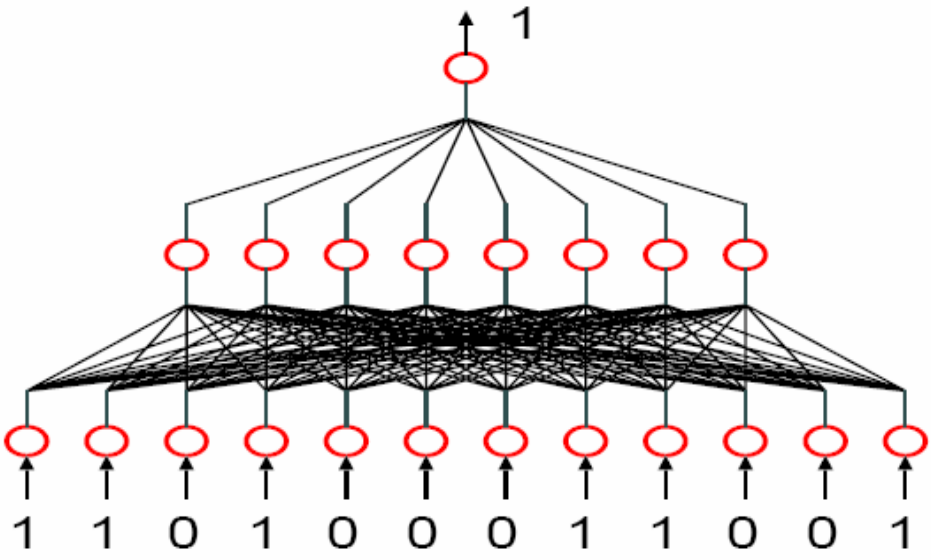
Рассмотрим нейронные сети большей размерности. Например, задача разделения N-мерных точек (N parity problem).

XOR

X_1	X_2	Y
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	+1







7 NP-полная Комбинаторная Задача Оптимизации

7.1 Задача о рюкзаке

Это одна из наиболее известных комбинаторных задач оптимизации.

Предположим, что имеется n предметов, которые можно положить в рюкзак. Каждый предмет имеет вес w_i и коэффициент полезности p_i. Далее для каждого i-го предмета подбираются неотрицательные целые значения x_i, где i=1,2,…,n. Цель заключается в поиске максимума для выражения:

$$\sum_{i=1}^{n} x_i p_i. \tag{4}$$

причём так, чтобы

$$\sum_{i=1}^{n} x_i w_i < C \tag{5}$$

где C – максимально возможный вес рюкзака.

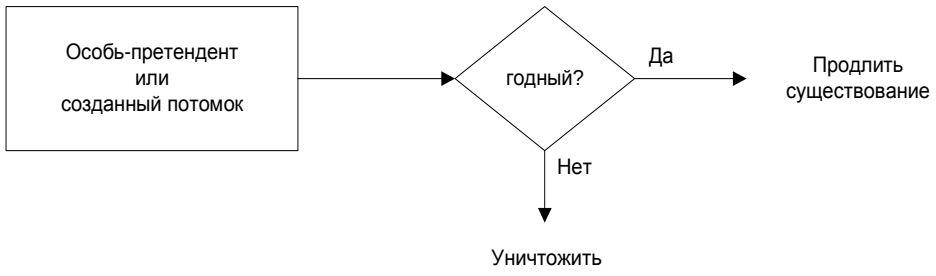
Применить генетический алгоритм в данном случае достаточно просто. Наша хромосома задается в форме:

$$(x_1 x_2 x_3 \cdots x_n) \tag{6}$$

где x_i - количество i-ых предметов, помещенных в рюкзак.

7.2 Удаление непригодных хромосом

Необходимо отметить что, если хромосома не удовлетворяет выражению (5), то такая хромосома удаляется, а процедура генерации хромосомы потомка (кроссовер, мутация и т.п.) повторяется снова до тех пор, пока не будет получена подходящая хромосома потомка.



8 Комбинаторная Оптимизация II

8.1 Линейная задача о назначениях (ЛЗН)

8.2 Квадратичная задача о назначениях (КЗН)

8.3 Задача коммивояжера (ЗК)

Предположим, что заданы координаты N городов. Задача коммивояжера (ЗК) заключается в том, что торговец должен посетить каждый из этих городов по одному разу, при этом его путь должен быть минимален.

Я нашел координаты 13 509 городов в США на сайте
http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95.
Почему бы не попробовать решить эту интересную задачу. Графическое представление всех этих городов на двумерной плоскости показано на Рисунке 9.

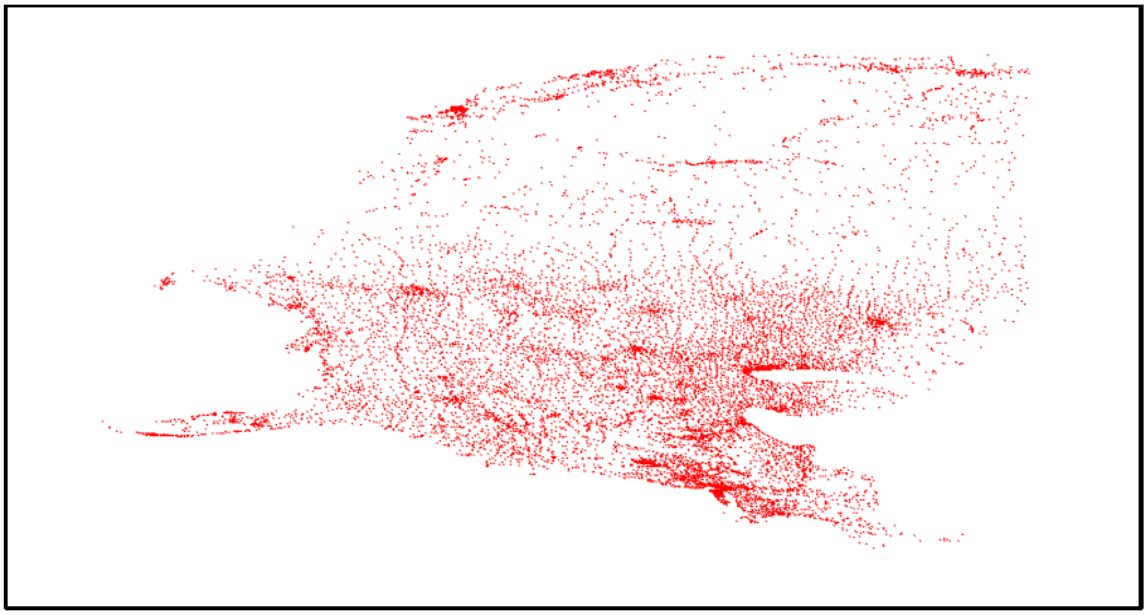


Рисунок 6: Координаты 13 509 городов США (использовались данные с http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95).

Вопрос заключается в том, как сформировать хромосому, задающую путь? Представим предполагаемое решение в виде списка городов, которые необходимо посетить в определенном порядке. В этом случае хромосома пути A-C-F-D-G-E-B будет иметь вид:

$$(ACFDGEB) \tag{7}$$

Применимы ли в этом случае результаты процедуры скрещивания и мутации? Ответ – Нет! Например, возможный потомок двух родителей (*ACFDGEB*) и (*AGBFCDE*) после одноточечного скрещивания (*one-point-crossover*) будет иметь вид (*ACFFCDE*). А такой случай не допустим, так как C и F встречаются дважды, а B не посещается вообще. Или, если применить стандартную процедуру мутации к (*ACFDGEB*), например, путем замены 4-го гена на другой случайно выбранный город - (*ACFAGEB*), то такой результат тоже является непригодным.

Тогда как правильно применить операции скрещивания и мутации?

Рассмотрим такой вариант решения этой проблемы:

Шаг-1. Установим $i = 1$.

Шаг-2. Если i-й ген равен n, тогда n-й город в списке является городом, который уже посещен.

Шаг-3. Удаляем город из списка.

Шаг-4. Установим $i = i + 1$ и повторяем пункты от Шага-2 до Шага-4 пока $i \leq n$.

Например, пусть список городов задан следующим образом:

{A, B, C, D, E, F, G, H, I}

хромосома: (112141311) задает путь:

A-B-D-C-H-E-I-F-G.

Попробуйте выполнить процедуру одноточечного скрещивания двух хромосом: (112141311) и (515553321).

Далее, как выполнить мутацию? Что если, например, случайным образом выбирать две точки и выстраивать гены между ними в обратном порядке?

И, наконец, когда же все-таки необходимо остановить алгоритм? Ведь, оптимальное значение нам не известно. В таком случае можно использовать значение оценки приспособленности. Мы можем предполагать, что результат будет сходиться к оптимальному значению или, как минимум, к значению близкому к оптимальному.

Таким образом, мы рассмотрели решение задачи коммивояжера при помощи генетического алгоритма. Но существует лучший алгоритм, известный как Оптимизация по Принципу Муравейника (Ant Colony Optimization - ACO). ACO – это механизм оптимизации, заимствованный у муравьев, и отражающий их коллективное поведение. Муравьи обладают способностью находить кратчайшие маршруты от места расположения еды до муравейника. Когда один муравей обнаруживает еду, он информирует других при помощи особого химического вещества, называемого феромон (pheromon). Если у нас будет достаточно времени, мы рассмотрим алгоритм ACO более подробно.

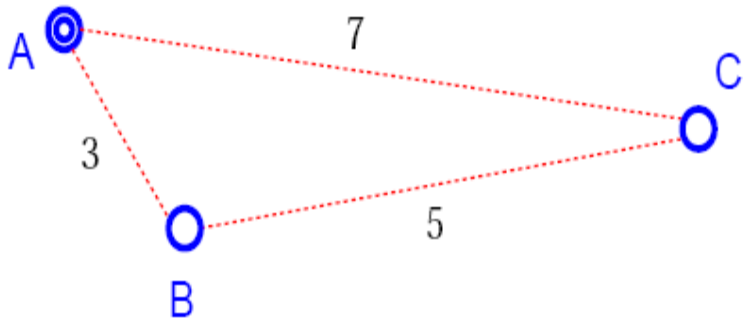


Рисунок 7: Пример с тремя городами. Здесь у нас только один возможный маршрут.

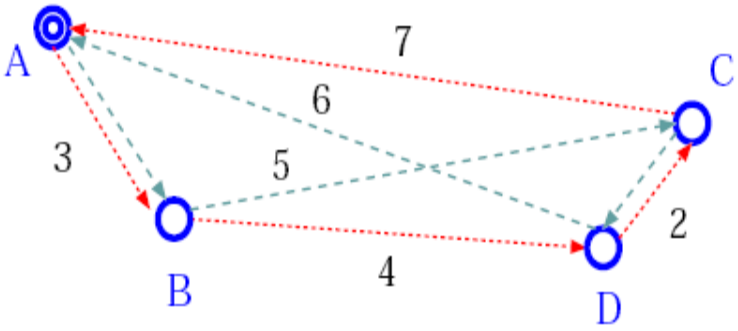


Рисунок 8: Пример с четырьмя городами. Тоже простой, но как вы считаете, какой маршрут будет самым коротким?

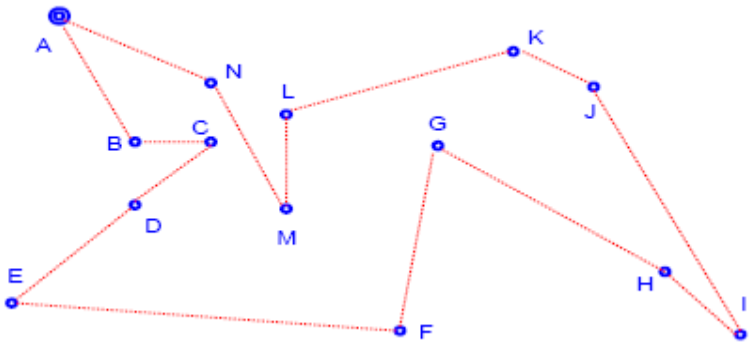


Рисунок 9: Пример с четырнадцатью городами и единственно возможный маршрут.

9 Перемещение робота в “клетчатом пространстве”

Предположим, что мы хотим запрограммировать агента или робота в клетчатом пространстве. В этом случае хромосома может состоять из целочисленных генов, принимающих значения 1, 2, 3 и 4, что соответствует перемещению агента на одну позицию в направлении север, юг, восток и запад соответственно. Пример приведен ниже:

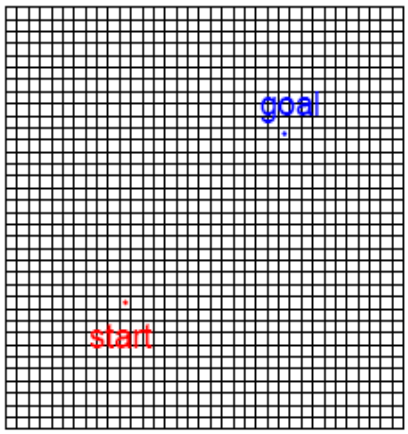


Рисунок 10: пример задачи поиска наикротчайшего пути в случае отсутствия препятствий.

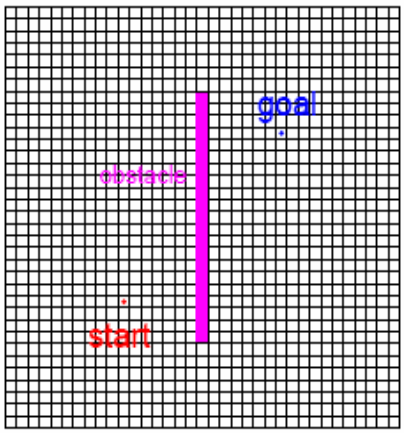


Рисунок 11: пример задачи поиска наикротчайшего пути с препятствием в виде стены.

Наша хромосома состоит из 4 различных генов: (i) двигаться вверх, (ii) вниз, (iii) вправо и (iv) влево. Рассмотрим пример:

(13333114114411141322422223)

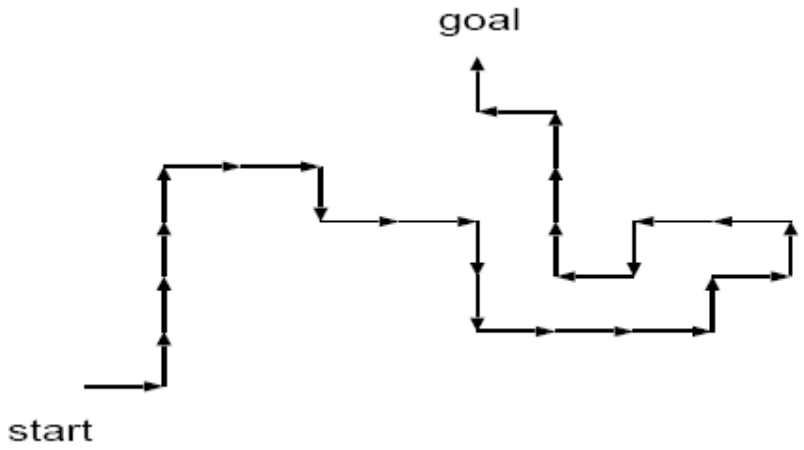


Рисунок 12: Пример хромосомы и заданного ею пути.

Рассмотрим две такие задачи.

9.1 Зондирование «клетчатого пространства» с ограниченной энергией

Поиск пути минимальной Манхэттеновской протяженности (Manhattan distance)

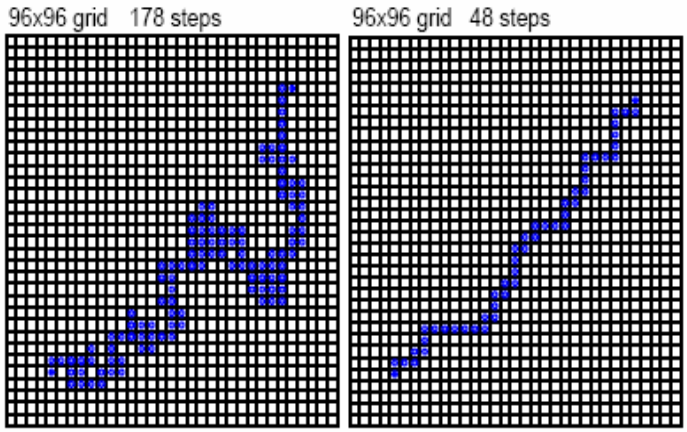


Рисунок 13: В пространстве размерностью 96x96 робот перемещается из клетки в позиции (24,24) в клетку (72,72), о которой ему заранее ничего не известно. Слева на рисунке представлен путь минимальной длины, сгенерированный случайным образом и отобранный из 100 экспериментов. Справа - путь минимальной длины, найденный роботом в результате применения эволюционного обучения, как показано на Рисунке 3. (Крайние области не показаны.)

Поиск пути к начальной точке после максимально возможного зондирования

Для человека определить такой маршрут не составляет особого труда. Но как быть в случае использования компьютера?
Вопрос заключается в том: “Как для этого случая задать функцию приспособленности?”

Рассмотрим его позже, в разделе, посвященном составным функциям приспособленности.

9.2 Задача Джипа
- от “Верблюда в пустыне” до “Марсохода”

Предположим, на некоторой базе на краю пустыни находится джип. Бензобак джипа может быть заполнен максимум одной порцией бензина. С одной порцией бензина джип может преодолеть некоторое единичное расстояние. Заправку джип может осуществить только на базе. Но джип может перевозить контейнеры в пустыню для размещения в них запасов бензина для последующего использования. [1]

Возникает вопрос: “Насколько далеко по прямой линии джип сможет заехать в пустыню, если на базе имеется n единиц бензина?”

Допустим $n = 2$, в данном случае лучшей стратегией будет следующая: выехать из базы с одной единицей бензина в баке, пройти 1/3 единицы расстояния (к этому моменту уже будет израсходована 1/3 бензина), оставить в этой точке контейнер с 1/3 бензина (у джипа еще в баке остается 1/3 бензина) и вернуться обратно на базу. Когда джип вернется на базу весь бензин, взятый в начале пути, будет израсходован. Далее джип заполнит бак второй единицей бензина и, проехав 1/3 единицы расстояния до оставленного в пустыне контейнера, дозаправится из него, и бак будет снова полон. Потом проедет вперед, пока весь бензин в баке не будет израсходован. Таким образом, максимальная дистанция, которую сможет преодолеть Джип, составит 4/3 единицы расстояния.

Мы знаем максимальную дистанцию для $n = 2$. Максимальную дистанцию D_n для n единиц бензина можно получить рекурсивно из выражения $D_n = (D_{n-1} + 1)/(2n - 1)$.

Для нас интерес представляет возможность поиска оптимальной или близкой к оптимальной стратегии при помощи эволюционных вычислений. Так для случая $n = 5$ максимальная дистанция составляет 1323/945=1.4. (Если мои вычисления верны. Проверьте самостоятельно.)

[1] Задача первоначально относилась к верблюду, перевозящему зерно через пустыню. Это была 52 задача в сборнике “Propositions ad acuendos inventes” (лат.), авторство приписывается Алкуину (Alcuin) из Йорка (ок. 732–804 до н.э.). В последствии рассматривалась аналогичная задача джипа в пустыне, а еще позже – марсохода.

10 Схема сортировки
– Каково число элементарных операций сравнения?

Кто умнее? – Человек или Компьютер?

При написании алгоритма, часто возникает необходимость отсортировать набор элементов, упорядочив их в соответствии с некоторым критерием. Как, например, выполнить задачу сортировки 16 целочисленных значений в порядке возрастания? Мы выберем и сравним два элемента массива. В случае, если порядок не удовлетворяет условию сортировки, поменяем их местами и т.д.

Алгоритм 1 (Алгоритм Сортировки) *Предположим, необходимо отсортировать N числовых элементов в порядке от меньшего к большему.*

For i = 1 to N-1
For j = i+1 to N
If item(i) < item (j) Then меняем item(i) and item(j)

Теперь представим вышеописанную сортировку графически:

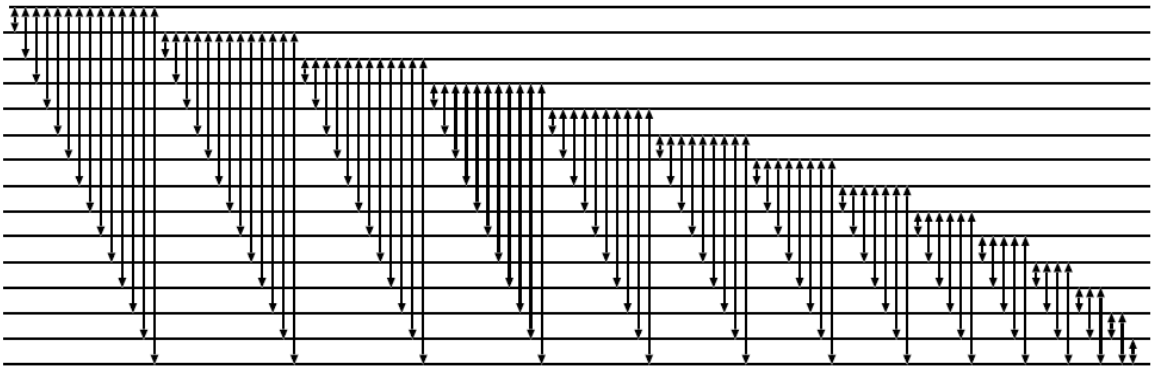


Рисунок 14: Типичная схема сортировки, но недостаточно эффективная.

Общее количество сравнений в данном случае 120, впрочем, это не самый эффективный метод.
Поэтому вопрос заключается в том, чтобы подсчитать минимальное число операций сравнения, при котором любой произвольный набор из 16 значений будет правильно отсортирован.

Задача (Схема Сортировки). *Нужно отсортировать n элементов. Для этого сравниваются значения i-го и j-го элемента и, при необходимости, элементы меняются местами. Цель заключается в поиске алгоритма, способного за минимальное количество операций сравнения отсортировать исходное множество из n элементов.*

Возможно, будет интересно ознакомиться с предысторией этого вопроса. В 1960-х было проведено соревнование на предмет поиска алгоритма, способного за минимальное количество операций сравнения обработать некоторый массив элементов (например, когда их количество равно 16, n=16). Были получены следующие результаты:

- 65 сравнений (Bose and Nelson, 1962).
- 63 сравнения (Batcher, Floyd and Knuth, 1964).
- 62 сравнения (Shapiro, 1969)
- 60 сравнений (Green 1969)

Смотрите рисунок ниже.

Batcher сортировка: 63 сравнения (Knuth 1973):

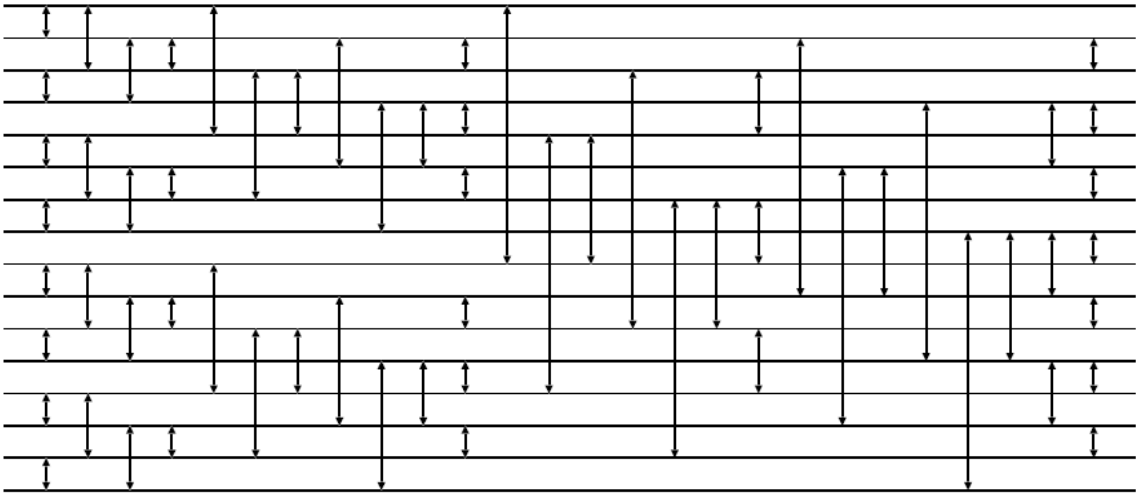


Рисунок 15: Предложенная Batcher'ом схема сортировки с 63 сравнениями (1964).

Однако, до сих пор не существует доказательства оптимальности этого решения. Тогда, давайте, попробуем применить Эволюционные Вычисления для поиска этого минимального числа операций. Окажется ли результат применения такого подхода эффективнее результата предложенного человеком? В 1992 году Hillis проводил свои исследования в этой области. Он использовал особую хромосому Диплоидия (*Diploidy Chromosome),* детальное описание которой дадим позже. Здесь же рассмотрим упрощенный вариант применения генетического алгоритма.

Предположим одна хромосома, соответствующая определенной схеме сортировки, состоит из 140 генов, каждый из которых принимает целочисленное значение от 1 до 16, допуская повторения, например:

(12 01 05 04 16 12 04 14 01 02 06 ..... 07 15 08 10)

здесь гены с нечетными номерами образуют пары с соседними справа генами с четными номерами, как показано ниже:

12 <=> 01; 05 <=> 04; 16 <=> 12; .....; 08 <=> 10

Таким образом, одна хромосома описывает максимум 70 операций сравнения. Почему максимум? Да потому что одни и те же пары могут встречаться несколько раз. Отсюда

следует, что минимальное количество операций сравнения равняется единице, хотя такая ситуация маловероятна.

11 Планирование стратегии – Итеративная Дилемма Заключенного

Итак, Быть или не быть? – Вот в чем вопрос. Не только работы Шекспира, но и многих других авторов посвящены данной дилемме. Опера Пуччини (Puccini) «Тоска» (1899) является типичным примером[2].

11.1 Когда возникает дилемма?

Допустим, *n* человек являются участниками следующей игры. Каждый находится в изолированной кабине, откуда не видит других участников и не может с ними общаться. Кабины оборудованы кнопкой. Все участники могут находиться в кабинах на протяжении одной минуты. Если никто не нажмет кнопку, то каждый из них получит 100$, с другой стороны, если кто-то нажмет кнопку, или несколько человек сделают это, то первый получит 10$, а остальные не получат ничего. Как бы поступили вы в этой ситуации?

11.1.1 Условия наличия дилеммы

Что если в случае, когда кнопка не будет нажата, каждому будет выдано только 10$, а в случае, если кнопка будет все-таки нажата, награда в 100$ достанется участнику, нажавшему ее первым. В такой ситуации никакой дилеммы не возникнет: нажимайте кнопку незамедлительно и даже не думайте.

В *Теории Игр* существует задача, которая называется Дилемма Заключенного[3] и формулируется следующим образом:

Задача (Дилемма Заключенного) *Двум арестантам A и B предлагается сделка:*

- *Если A сознается, а B нет, то A будет освобожден, а B получит 5 лет тюрьмы, и наоборот.*
- *Если сознаются оба, тогда оба получат по 4 года тюремного заключения.*

---

2 Matt Ridley □ □□□□□ □□□□□ □□□□□□□□, □□□□□□□□□□ □□□□ □□□□□□□□□□ - □□□□□□□□□□□□□ □□□□□□□□□□ □ □□□□□□□□ □□□□□□□□□□. □ □□□□□□ □Penguin Books□ (1996) про эту оперу сказано следующее: “Героиня оперы Пуччини “Тоска” столкнулась со сложнейшей дилеммой. Ее возлюбленный Каварадосси был приговорен к смертной казни шерифом по имени Скарпиа. Однако шериф предложил ей сделку. Если Тоска переспит с ним, то он сохранит жизнь возлюбленного, приказав использовать холостые патроны. Тоска решает обмануть Скарпиа. Она соглашается на его предложение, но надеется убить его, после того как будет дано распоряжение об использовании холостых патронов. Она выполняет свой план, но только после узнает, что Скарпиа тоже обманул ее. Каварадосси мертв, распоряжения об использовании холостых патронов так и не последовало. Тоска кончает жизнь самоубийством. В итоге все три героя оперы мертвы”. Эту историю можно рассмотреть с позиции Теории Игр. Автор пишет: “Что если бы все было иначе? Тоска и Скарпиа являются участниками одной игры, самой распространенной в теории игр – особой области математики, представляющей связь между биологией и экономикой. Эта игра является предметом особого интереса научных исследований последнего времени: ничего более, чем просто понимание того, почему люди нравятся друг другу. Кроме того, Тоска и Скарпиа сыграли свои партии в этой игре так, как это и должно было получиться в соответствии с Теорией игр, несмотря на печальный исход игры. Как это понимать?”

3 Предложена Меррил Флудом (Merrill Flood) и Мелвин Драшером (Melvin Dresher) в 1950-х

- *Если оба будут молчать, тогда каждый получит по 2 года.*

Очевидно, что наилучший исход – это “0 лет тюрьмы”, следующим лучшим решением будет “2 года”, далее следуют “4 года” и “1 год”, как наихудшие варианты развития событий, если же вы, конечно, не считаете тюрьму “бесплатной гостиницей”.

Далее сопоставим рассмотренным исходам числовые значения 15, 10, 6 и 0 соответственно. Тогда:

	A получит	B получит
A сознается & B молчит	15	0
A сознается & B сознается	6	6
A молчит & B молчит	10	10
A молчит & B сознается	0	15

В таком случае A может получить ”15” с возможным риском получить “6”, но A может выбрать “10”, что больше чем “6”, опять же с риском получить “0” в наихудшем случае. Здесь возникает дилемма.

Рассмотрим другой вариант. Возникнет ли дилемма в данной ситуации?

	A получит	B получит
A сознается & B молчит	15	0
A сознается & B сознается	10	10
A молчит & B молчит	6	6
A молчит & B сознается	0	15

Ответ “Нет”. Заключенный A сознавшись может получить “15” (при благоприятном развитии событий) или 10 (при неблагоприятном развитии событий), тогда как промолчав – “6” или “0”. Таким образом, дилеммы нет. Сознавайтесь немедленно! Независимо от реакции оппонента такая стратегия более выгодна, чем просто молчание.

	A получит	B получит
A сознается & B молчит	10	1
A сознается & B сознается	3	3
A молчит & B молчит	6	6
A молчит & B сознается	1	10

Все-таки, при каком условии возникает дилемма? Предположим:

	A получит	B получит
A сознается & B молчит	γ_3	γ_2
A сознается & B сознается	γ_4	γ_4
A молчит & B молчит	γ_1	γ_1
A молчит & B сознается	γ_2	γ_3

Из двух рассмотренных выше примеров следует

$$\gamma_3 > \gamma_1 > \gamma_4 > \gamma_2. \tag{8}$$

	A получит	B получит
A сознается & B молчит	15	0
A сознается & B сознается	3	3
A молчит & B молчит	6	6
A молчит & B сознается	0	15

Как насчет такого случая?

Это следует из вышеприведенного условия. Но возникает ли дилемма в этом случае? Видимо, “Нет”. Поэтому необходимо другое условие:

$$2\gamma_1 > \gamma_2 + \gamma_3 \tag{9}$$

Если вы не согласны, то представьте более экстремальную ситуацию:

	A получит	B получит
A сознается & B молчит	15	0
A сознается & B сознается	1	1
A молчит & B молчит	2	2
A молчит & B сознается	0	15

Как вы видите, молчание не выгодно.

Хотя дилемма не столь сложна, как в случае с Гамлетом (“Быть или не быть? Вот в чем вопрос?”), тем не менее, всегда лучше сознаваться.

Если заключенные вовлечены в итеративную игру, то такую ситуацию нужно рассматривать иначе. Как и на любых переговорах, здесь имеет место дилемма – сотрудничать или предавать?

11.1.2 Итеративная Дилемма Заключенного

Но как быть, если игра повторится и, возможно, несколько раз? Такая ситуация известна как *Итеративная Дилемма Заключенного (Itereted Prisoner’s Dilemma - IPD)*. В такой игре выбирается стратегия, которая в результате позволит получить наибольшую выгоду (награду). Но какое действие выбрать на определенном этапе игры? Например, можно остановиться на стратегии типа «Всегда Предательство» или стратегии «Услуга за Услугу», в которой игроки сотрудничают в первой игре, а в дальнейшем просто повторяют шаги оппонента в предыдущей игре.

Рассмотрим ситуацию, когда стратегия выбирается на основании трех предшествующих шагов двух игроков. Количество всевозможных вариантов трех предшествующих игр составляет $2^6 = 64$ — 64 комбинаций Сотрудничества и Предательства. Все эти возможные комбинации 6 предыдущих шагов могут быть представлены 64-битной хромосомой. Например, пусть история 6 шагов некоторого игрока и его оппонента задана в виде C-d-D-d-C-d. Тогда в бинарном представлении эта последовательность задается следующим двоичным числом - 100010, где “C” и “c” соответствуют 1, “D” и “d” соответствуют 0, прописные “C” и “D” - действия оппонента, строчные “c” и “d” - действия игрока, C – “Сотрудничество”, D – “Предательство”.

Таким образом, первая бинарная последовательность задается в виде (000000). Для получения следующей последовательности устанавливаем первый бит в 1, что означает Сотрудничество. Теперь, когда имеется история (000001), меняем второй бит – и получаем (000010), третий бит и т.д. Повторяем подобные действия пока не будет получен последний вариант – (111111). Используя этот нехитрый алгоритм можно без особого труда перечислить все 64 возможные комбинации.

Далее для каждой такой последовательности (хромосомы) случайным образом выбираются p хромосом, и подсчитывается количество побед рассматриваемой хромосомы над отобранными. Полученное значение количества одержанных побед используется для отбора оптимальной стратегии (т.н. р- турнирная селекция).

12 Визуализация многомерного пространства

Порой очень важно наглядно отобразить информацию, заданную в многомерном пространстве. Вполне возможно вы уже знакомы с такими методами как *Самоорганизующиеся Карты Кохонена (Kohonen's Self Organizing Map - SOM)* или *Метод Главных Компонент (Principal Component Analysis - PCA).*

12.1 Почему необходимо понижать размерность?

Люди не в состоянии представить мир в более чем трехмерном пространстве. Однако во многих сферах науки особенно важно воспринимать картину пространства высокой размерности. Но помимо науки такая задача часто возникает и в повседневной жизни.

Приведем пример. Допустим, необходимо определить специализацию для каждого вновь прибывшего солдата в зависимости от его способностей к Математике и Английскому языку.

	A	B	C	D	E	F	G	H	I	J	K	L	M
Математика	95	32	89	52	12	20	3	99	42	91	26	95	60
Английский	92	90	21	48	14	5	11	97	50	92	89	13	55

Таблица 1: Результаты новобранцев по двум экзаменам.

Задача классификации солдат станет проще, если полученные на экзаменах результаты отобразить графически.

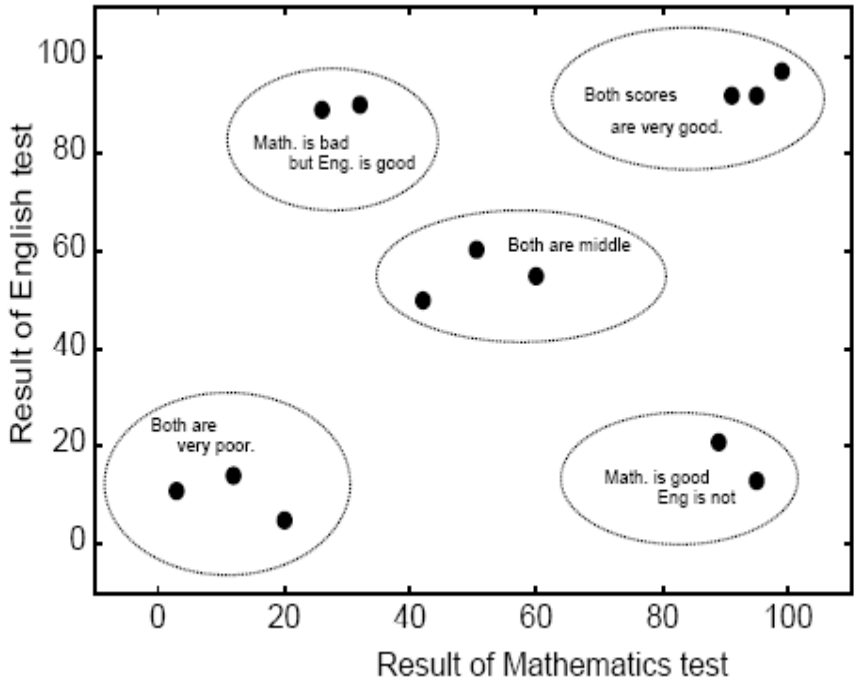


Рисунок 20: Результаты визуализации. Достаточно просто классифицировать солдат на 5 категорий.

Но как быть, если у нас имеются оценки еще по одной дисциплине, например, по физической подготовке. Так для разведки хорошая физическая подготовка является очень

полезным качеством. В таком случае нам необходимо рассматривать трехмерные данные, и отображать их придется в трехмерном пространстве. Усложним задачу, представив, что оценки выставлены по 10 экзаменам. Визуализировать в обычном виде такой набор данных уже невозможно.

Таким образом, визуализация многомерного пространства, а также задача сокращения размерности очень важны. Было предложено большое количество методов по этой тематике, и Самоорганизующиеся Карты Кохонена является одним из самых широко распространенных.

12.2 Отображение Sammon при помощи генетического алгоритма

В данном разделе рассматривается т.н. Отображение Sammon. Отображение Sammon – это отображение набора точек многомерного пространства в двумерное пространство с насколько это возможным сохранением соотношения расстояний между элементами исходного пространства. Другими словами, задача состоит в аппроксимации расстояний исходного n-мерного пространства соответствующими расстояниями в 2-мерном пространстве с минимально возможными потерями.

Метод был предложен в 1980-х в качестве задачи оптимизации, к которой был применен не самый простой алгоритм Наискорейшего Спуска из области Исследования Операций. С другой стороны применить Эволюционные Вычисления в этой ситуации гораздо проще. Разберемся теперь, что представляет собой Отображение Sammon:

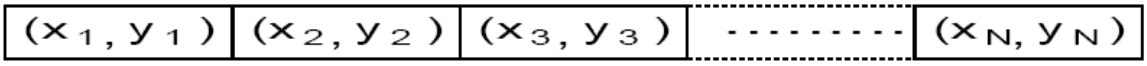
Алгоритм (Отображение Sammon)

1. Допустим, заданы N точек в n-мерном пространстве.
2. Рассчитаем матрицу расстояний R (N X N), где элемент в позиции i-j – Евклидово расстояние между i-ой и j-ой точкой.
3. Определим также N точек в двумерном пространстве и для начала распределим их случайным образом.
4. Высчитаем матрицу расстояний Q, аналогично матрице R.
5. Рассчитывается матрица ошибок, как $P = R - Q$.
6. Осуществляется поиск позиций N точек в двумерном пространстве таким образом, чтобы минимизировать суммарное значение элементов матрицы P.

Таким образом, мы имеем дело с задачей оптимизации, которая, как нам известно, может быть легко решена при помощи Эволюционных Вычислений. Наша задача *заключается в поиске N точек в 2-мерном пространстве, которые соответствовали бы N точкам в n-мерном пространстве. При чем при отображении необходимо сохранить, на сколько это возможно, все соотношения расстояний между точками исходного n-мерного пространства, т.е. выполнить аппроксимацию с минимальной ошибкой.*

При реализации генетического алгоритма для решения поставленной задачи хромосомы должны включать *n* генов, каждый из которых соответствует искомой координате *x-y* точки в 2-мерном пространстве. Применяется операция равномерного скрещивания, а время от времени – мутация, замещающая один ген случайной координатой x-y (см. Рисунок 2, см. Рисунок представленный ниже).
Пример для $49^2 = 2401$ мерного пространства:

Хромосома:



Равномерное Скрещивание:

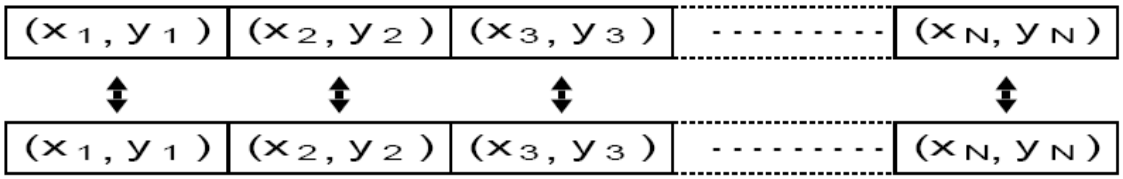


Рисунок 17: Представление хромосомы и равномерное скрещивание.

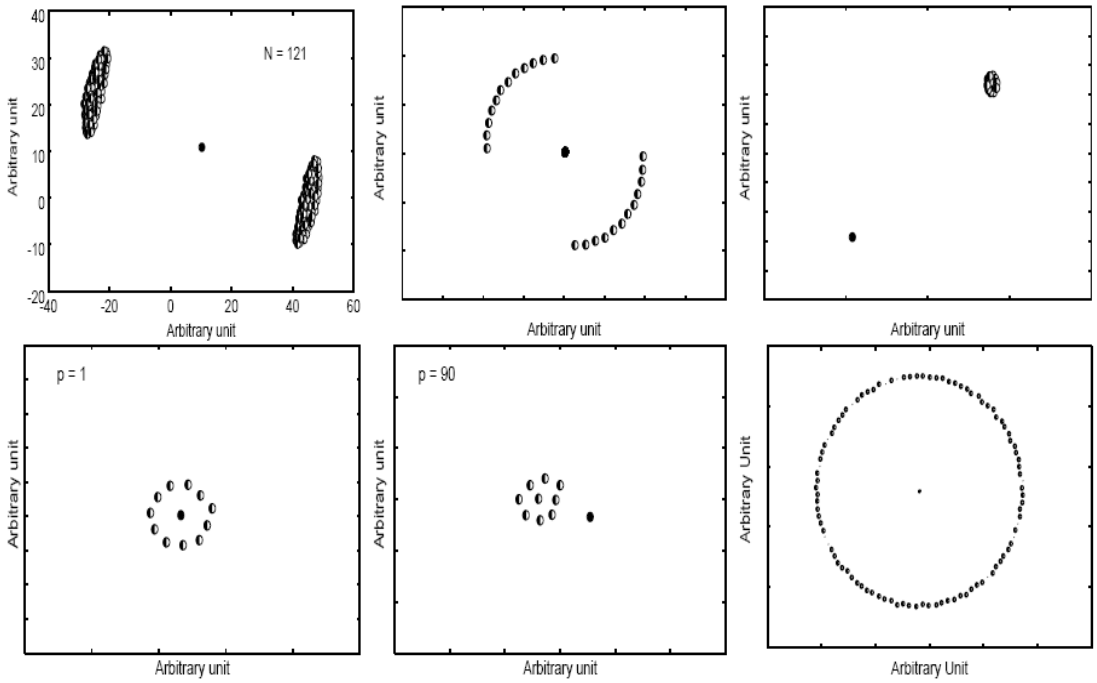


Рисунок 18: Шесть примеров Отображения из 2401-мерного пространства в 2-мерное пространство. Разъяснения приведены в тексте.

13 Возвращение к схемам сортировок

13.1 Больше биологии — использование хромосом Диплоидий

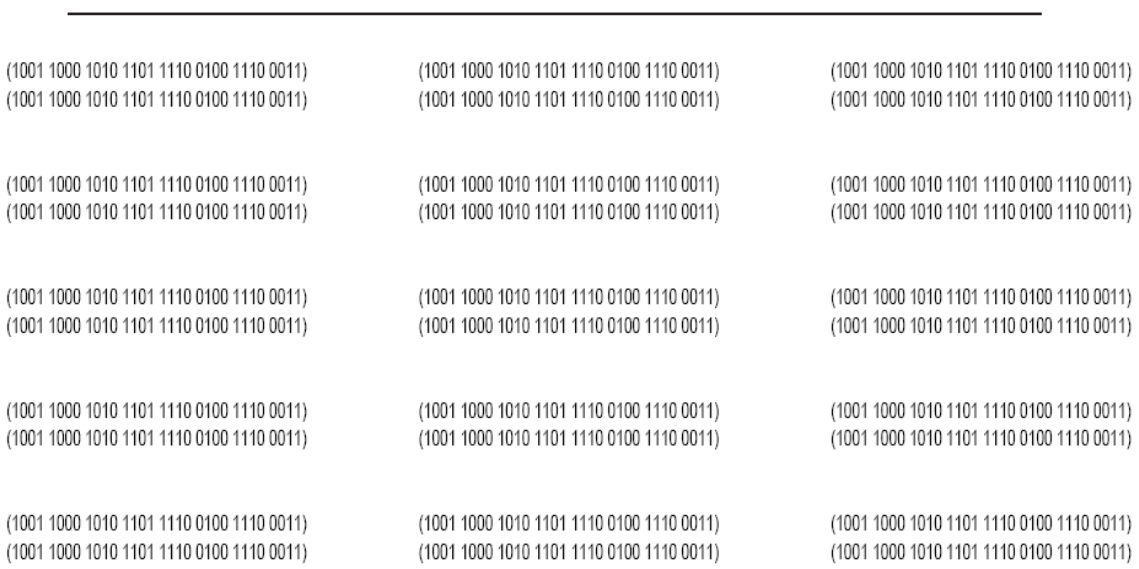


Рисунок 19: Пример набора хромосом Диплоидий, подготовленных Hillis-ом.

- Каждая особь состоит из 15 пар 32-битных хромосом.
- Каждая хромосома состоит из восьми 4-битных строк (называемых *кодонами*).

(0001 0010 0101 1000 0000 0100 1111 1001)
(0011 0100 0101 1000 1101 1100 1111 1001)

- Каждый кодон представляет собой целое число в диапазоне от 0 до 15, указывающее какой из 16 элементов будет принимать участие в сравнении. Для вышеприведенного примера:

(01 02 05 08 00 04 15 09)
(03 04 05 08 13 12 15 09)

- Каждая пара соседних кодонов в хромосоме определяет операцию сравнения между двумя элементами. Таким образом, каждая хромосома кодирует четыре операции сравнения, например:

(09 08 10 13 14 04 14 03)

определяет четыре следующих операции сравнения.

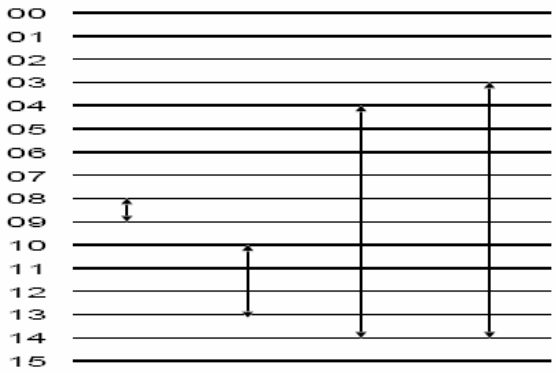


Рисунок 20: Четыре сравнения, заданные хромосомой (09 08 10 13 14 04 14 03).

- Пары кодонов считываются слева направо.

- В случае если два соседних кодона совпадают с кодонами в той же позиции парной хромосомы (*гомозигота*), то только одна пара чисел помещается в фенотип. Если соответствующие пары кодонов кодируют различные пары чисел (*гетерозигота*), тогда обе пары чисел помещаются в фенотип. Так для вышеприведенного примера:

  (01 02 05 08 00 04 15 09)
  (03 04 05 08 13 12 15 09)

  сформированы шесть операций сравнения:

  01<=>02, 03<=>04, 05<=>08, 00<=>04, 13<=>12, 15<=>09

- Получается, что 15 пар хромосом образуют фенотип с числом операций сравнения заданным в пределах от 60 до 120. Чем чаще встречаются гомозиготы, тем меньше операций сравнения.

- После того как отобраны две особи, выполняется одноточечное скрещивание для каждой пары хромосом в каждой особи.

- Для каждой из 15 пар хромосом позиция скрещивания выбирается случайно и образуется отдельная хромосома (называемая гамета).

- В результате для каждой родительской особи формируется по 15 гамет.

- Каждая из 15 гамет от одного родителя образует пару с соответствующей гаметой второго родителя, формируя тем самым потомка.

13.2 Стремление к гомозиготным парам

У гомозиготных пар вероятность выжить больше чем у гетерозиготных пар, поскольку очевидно, что для двух одинаковых генов в одной и той же позиции хромосомы вероятность после эволюции сформировать гомозиготную пару выше.

Например, вероятность пары (1,1) остаться парой (1,1) составляет ½, в то же время вероятность (1,0) остаться (1,0) составляет ¼. Первое значение рассчитано по формуле

$$1 \text{ x } ((1/4) \text{ x } (1/2) + (1/4) \text{ x } (1/2) + (1/4) \text{ x0}),$$

в то время как второе –

$$(1/2) \text{ x } ((1/4) \text{ x } (1/2) + (1/4) \text{ x } (1/2) + (1/4) \text{ x0}).$$

Отсюда следует, что в результате длительного эволюционного процесса количество гомозиготных генных пар увеличится. В нашем случае, если в результате вычислений каждая пара будет состоять из одинаковых хромосом, то это будет означать, что количество операций сравнения равно 60.

14 Конечные Автоматы и Эволюционные Вычисления (Finite State Machine - FSM)

14.1 Что такое FSM?

Человек постоянно подвержен влиянию различных чувств, которые формируют его СОСТОЯНИЕ. Оно в свою очередь постоянно меняется под воздействием информации поступающей от сенсоров (например, глаз). Результатом оценки и обработки поступающей информации является ДЕЙСТВИЕ. Приведем пример, когда я пребываю в подавленном СОСТОЯНИИ, а от одного из моих сенсоров “рот” поступает ВХОДНОЙ сигнал “водка”, то мое СОСТОЯНИЕ меняется на “счастье”, и результатом будет ДЕЙСТВИЕ – “пой песни”.

Аналогичным образом можно представить работу Конечного Автомата (КА), который характеризуется тремя составляющими: состояние, входные данные, действие. Работа КА начинается с определенного состояния. Под воздействием входных данных происходит изменение состояния и выполняется некоторое действие. Это один шаг. Такая последовательность действий

(1) изменение состояния в соответствии с поступающими на вход данными;
(2) реакция в виде выполнения некоторого действия

повторяется снова и снова, формируя некоторую модель поведения КА.

14.2 Задание в качестве примера

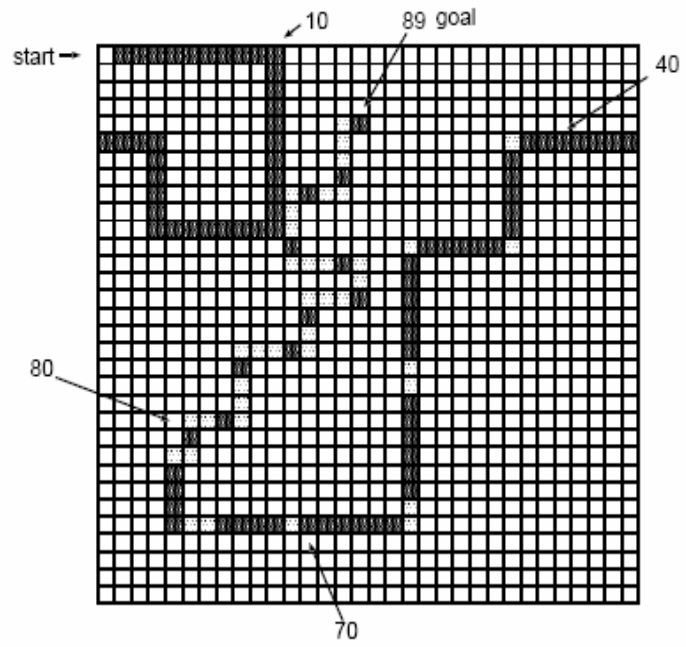


Рисунок 21: След Джона Мура на матрице 32 × 32.

14.2.1 Метод с использованием Нейронной Сети

14.2.2 Метод с использованием Конечного Автомата

Пример использования КА с 4-мя состояниями
КА использует следующую стратегию для решения выше обозначенной задачи: двигаться вперед, если перед ним ячейка со следом; если он не видит перед собой след, он разворачивается вправо (не перемещаясь из занятой ячейки) и проверяет наличие следа в этом направлении. Если след обнаружен, он продолжает движение вперед, но если следа нет, он снова поворачивается вправо. КА в поисках следа может повернуться до 4 раз. После этого он примет исходное направление и выполнит продвижение вперед, даже при отсутствии следа. После этого он опять займется поиском следа по 4-ем направлениям.

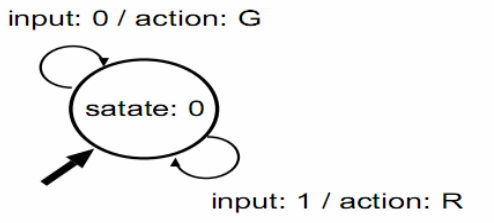


Рисунок 22: Пример простейшего КА.

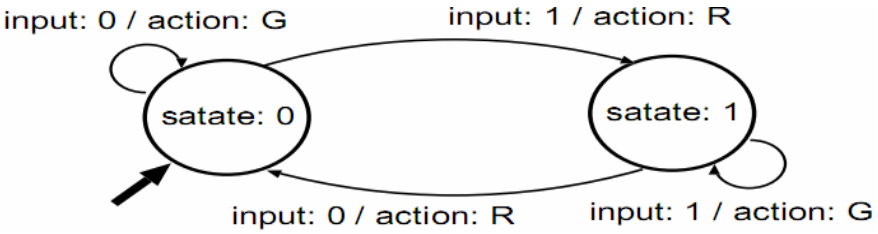


Рисунок 23: Еще пример КА. Как он функционирует?

Таблица 2: Пример того, как представить таблицу переходов в виде хромосомы.

Old State	Input	New State	Action
00	0	00	01
00	1	01	11
01	0	01	01
01	1	11	11
10	0	10	10
10	1	11	11
11	0	10	10
11	1	10	11

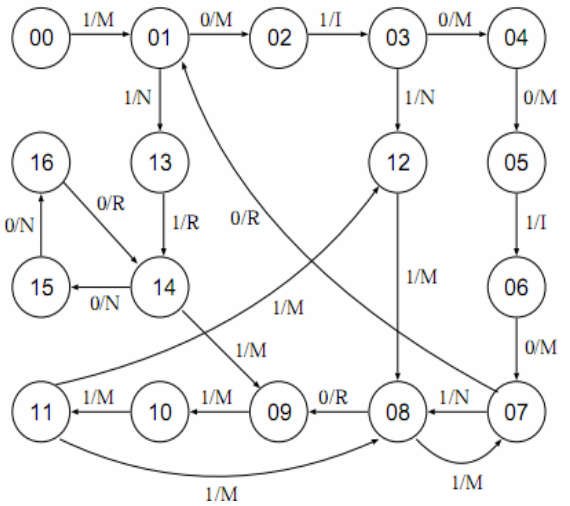


Рисунок 24: Результат эволюции КА.

14.2.3 Скрытый Марковский Процесс (Hidden Markov Process - HMM)

В отличии от КА, который является детерминированным методом, Скрытый Марковский Процесс (СМП) является недетерминированным.

15 Мультимодальная проблема

Как быть, если у нас имеется множество значимых решений?

В задаче Коммивояжера нам было важно найти только самый короткий путь из множества полученных решений. Однако иногда нас интересуют все возможные решения, полученные в процессе вычислений. Например, когда мы хотим получить множество нечетких правил для того, чтобы спроектировать нечеткий контроллер. Тема данного раздела посвящена этой проблеме. Давайте начнем с простых математических функций.

15.1 Еще одна функция

Двумерная функция с большим количеством пиков

Вопрос в том, как сформировать хромосому. В случае многомерной функции $y = f(x1, x2, \cdots, xn)$ каждый ген может соответствовать переменной x_i ($i=1, 2, \cdots n$) функции, таким образом, мы имеем n генов в хромосоме. Из-за того, что у нас имеется достаточное количество генов n, проблем не возникает. С другой стороны, как нам поступить в ситуации, когда в наличии только один ген. Использовать хромосому из одного гена? А как тогда выполнить операцию скрещивания?

Хорошо, в такой ситуации мы обычно используем хромосомы в бинарном представлении. Любое (десятичное) реальное значение переменной $x_i \in [a, b]$ может быть закодировано n-битной двоичной последовательностью, где a и b представлены соответственно (00 $\cdots$ 0)и (11 $\cdots$ 1). Точность вычислений при использовании такого подхода можно оценить по формуле $(b-a) / (2^n-1)$. Например, если наше значение принадлежит диапазону $x \in [0, 1]$, то при использовании 10-ти бит можно задать битовые строки от 0000000000 и до 1111111111. В этом случае точность десятичных чисел будет составлять 1/1024. Кроме того, вы можете воспользоваться *Кодом Грея*, где код-грея $a_1a_2\ldots a_n$ получается из двоичного числа $b_1b_2\ldots b_n$, по следующему правилу

$$a_i = \begin{cases} b_i & \text{if} \quad i = 1 \\ b_{i-1} \oplus p_i & \text{otherwise} \end{cases} \tag{10}$$

где $\oplus$ - сумма по модулю 2. В коде Грея расстояние Хемминга для пары смежных десятичных чисел отличается на 1, что в общем случае не выполняется для стандартной бинарной кодировки.

Функции, рассмотренные нами в разделе 4, особенно хорошо подходят для выполнения эволюционных вычислений, так как мы можем легко применить их к данным большой размерности, просто увеличив количество генов в хромосоме.

Но как быть в случае, когда задана 2-мерная функция? Например,

$$y = \sin^6(5\pi x) \tag{11}$$

или

$$y = -2((x - 0.2)/0.8)^2 \sin^6(5\pi x) \tag{12}$$

Эти функции представляют для нас особый интерес, поскольку мы можем исследовать на их примере поведение популяции хромосом, эволюционирующих в направлении поиска пиков (см. Рисунок 2).

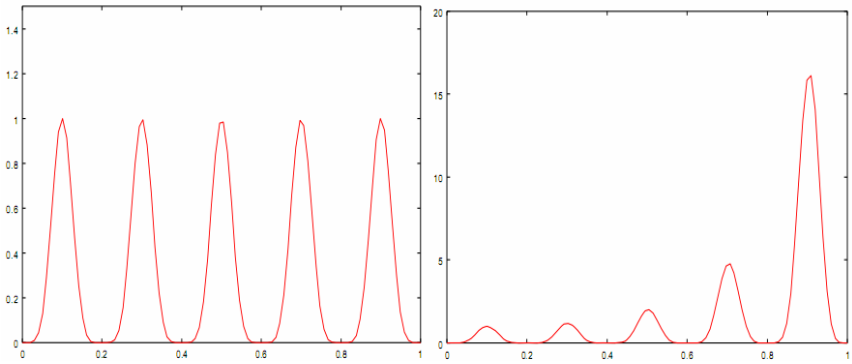


Рисунок 25: Мультипиковая 2-мерная функция и ее разновидность.

15.2 Мультимодальная оптимизация

Порой мы имеем множество решений. Но эволюционные вычисления зачастую сходятся только в одном из них. Поэтому для получения всех решений нам приходится выполнять алгоритм несколько раз. В этом подразделе будет рассмотрен метод, в котором формируются своеобразные ниши для каждого вида особей, представляющих отдельное решение.

Для этого, особи популяции разделяются на несколько *видов*. Каждый вид независимо от особей других видов выполняет поиск пиковых значений (в надежде обнаружить оптимальное решение), формирует свою *нишу* около некоторого пика и придерживается ее на протяжении остального времени выполнения программы.

Некоторое время назад, как правило, использовались следующие три метода. Это:

- Fitness sharing (Goldberg & Richardson, 1987)
  Схожие особи делят оценку приспособленности друг с другом.
- Crowding (De Jong, 1975)
  Схожие особи замещаются случайными особями
- Species Method
  Скрещивание ограничивается кругом схожих особей.

На сегодняшний день, по моему мнению, наиболее популярны следующие два метода:

- Detterministic Crowding (Mahfoud, 1992)
- Sequential Niching

Рассмотрим более подробно каждый из этих методов.

Fitness sharing: Приспособленность каждой особи снижается в зависимости от количества схожих особей в популяции. Это означает, что долевая оценка приспособленности $F_s(i)$ для i-ой особи рассчитывается по формуле

$$F_s(i) = \frac{F(i)}{\sum_{j=1}^{\mu} s(d_{ij})}$$

где *F(i)* – оценка приспособленности *i*-ой особи; d_{ij} – расстояние между *i*-ой и *j*-ой особью; обычно d_{ij} задается *расстоянием Хемминга* (в случае генотипического пространства) или *Евклидовым расстоянием* (в случае фенотипического пространства) и *s(·)* называется *функцией разделения*, которая задается:

$$s(d_{ij}) = \begin{cases} 1 - (d_{ij}/\sigma_{\text{share}})^{\alpha} & \text{if } d_{ij} < \sigma_{\text{share}} \\ 0 & \text{otherwise} \end{cases}$$

где σ_{share} - интерпретируется как размер ниши, а α - определяет форму функции. Этот знаменатель называют еще *отсчетом ниши*. На Рисунке 26 можно видеть, как форма $s(d_{ij})$ зависит от значения α.

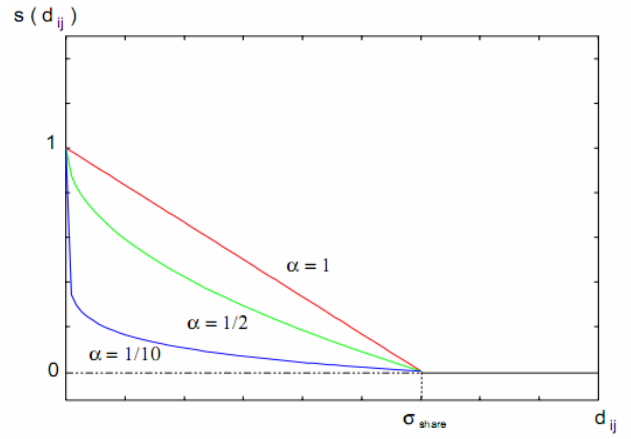


Рисунок 26: Зависимость формы $s(d_{ij})$ от α.

Проще говоря, схожие особи делят оценку приспособленности. Количество особей, которые концентрируются около некоторой вершины (ниши), ограничено. Теоретически их количество должно быть пропорционально *высоте* пика.

Detterministic Crowding: в этом случае вопрос о замещении родителей потомками решается в зависимости от значения расстояния между ними.

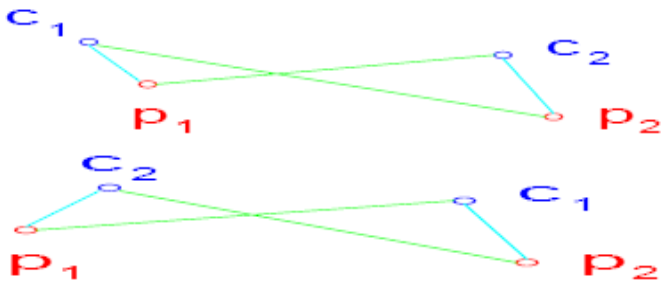


Рисунок 27: Два типичных примера расстояний между родителями и потомками.

Алгоритм. Предположим, что операции скрещивания, мутации, а также функция приспособленности уже заданы.

1. *Случайным образом выбирается пара родительских особей, p_1 и p_2, при чем ни один родитель не может быть выдран более одного раза.*
2. *Генерируются два потомка c'_1 и c'_2.*
3. *Выполняется мутация и скрещивание потомственных особей. В результате получаем c_1 и c_2.*
4. *Замещение родительской особи потомком происходит по следующим правилам:*

- *IF* $d(p_1, c_1) + d(p_2, c_2) > d(p_1, c_2) + d(p_2, c_1)$
  - *IF* $f(c_1) > f(p_1)$ *THEN* *replace* p_1 *with* c_1
  - *IF* $f(c_2) > f(p_2)$ *THEN* *replace* p_2 *with* c_2
- *ELSE*
  - *IF* $f(c_2) > f(p_1)$ *THEN* *replace* p_1 *with* c_2
  - *IF* $f(c_1) > f(p_2)$ *THEN* *replace* p_2 *with* c_1

где d (ζ_1, ζ_2) – расстояние Хемминга между двумя особами (ζ_1, ζ_2). Формирование потомков продолжается до тех пор, пока все особи популяции не примут участие в этом процессе. Далее те же действия по воспроизводству новой популяции и поиску циклически повторяются до тех пор, пока не будет найдено оптимальное решение или не будет превышено заданное количество популяций.

Sequential Niching: На каждом цикле выполнения алгоритма выбирается наилучшая особь.

Алгоритм:

1. *Определяется ниша радиусом r.*
2. *Составляется модифицированная функция приспособленности m(x), в результате присваивания исходной функции приспособленности f(x) (см. выше).*
3. *После выполнения генетического алгоритма выбирается наилучшая особь.*

4. *Пересчитывается значение m(x) по формуле*[4]

$$m_{n+1}(\mathbf{x}) = m_n(\mathbf{x}) \cdot g(\mathbf{x}, \mathbf{s}_n) \tag{14}$$

где n - число выполненных циклов алгоритма, s_n - лучшая особь, отобранная после n-ого цикла и

$$g(\mathbf{x}, \mathbf{s}_n) = \begin{cases} (d_{xs}/r)^{\alpha} & if\ d_{xs} < r \\ 1 & otherwise \end{cases} \tag{15}$$

является функцией понижения значения, в которой d_{xc} – расстояние между x) и s_n). Под m_0 понимается исходная не модифицированная функция приспособленности каждой особи.

5. *Выполняется генетический алгоритм с использованием модифицированной функции приспособленности и выбирается наилучшая особь.*
6. *Обновляется модифицированная функция приспособленности.*
7. *Если не модифицированная оценка приспособленности выбранной особи превосходит заданный порог (см. ниже), то получено окончательное решение.*
8. *Если не все решения найдены, то возвращаемся к шагу 5 алгоритма.*

[4] Это называется Степенная Функция Понижения. Альтернативой данной функции может служить Экспоненциальная Функция Понижения:

$$g_e(x, s) = \begin{cases} \exp((\log m(x, s)) \cdot (r - d_{xs}/r)) & \text{if } d_{xs} < r \\ 0 & \text{otherwise} \end{cases}$$

(13)

- Под порогом понимается

  В том случае, когда известно количество вершин, можно поэкспериментировать, понизив предел оценки приспособленности. В ином случае устанавливайте порог равным 0.

Задание: *Подобно рисунку 15.2, постройте график функции* $y = (x/r)^{\alpha}$ *, где* $r = 1$ *и* $\alpha = 0.5, 1, 2, 4, 8$. *Это позволит продемонстрировать, как выглядит* $g(x, s_n)$.

Так же будет интересно поэкспериментировать с мультимодальными эволюционными вычислениями для следующих двух функций:

$$(1)\ \ y = \sin^6(5\pi x)$$
$$(2)\ \ y = -2((x - 0.2)/0.8)^2 \sin^6(5\pi x)$$

16 Многоцелевой Генетический Алгоритм

До сих пор мы рассматривали как получить возможное решение(-ия) для одной целевой функции, допустим, максимизируя значение функции приспособленности. Однако в реальных задачах, как правило, нас интересуют несколько целей или критериев одновременно.

Часто цели противоречат друг другу. Например “время” и “деньги”: чем больше мы хотим зарабатывать, тем меньше остается времени тратить деньги; или ”надежность” продукции и ее “цена” при фабричном производстве. Или, предположим, необходимо подобрать певца для партии сопрано в опере. Критерием выбора являются: красота голоса, стройность фигуры, владение языками (итальянский, немецкий и т.д.). Увы, Бог не сделал нас талантливыми во всем.

Когда мы имеем дело с многоцелевой функцией, то в первую очередь должны разработать саму стратегию получения оптимального решения удовлетворяющего множеству функций или, другими словами, “недоминируемого решения”.

Определение: под “недоменируемым” или “Паретто-оптимальным” решением *понимается такое решение, которое является наилучшим с учетом всех целевых функций.*

Предположим, мы имеем *n* целевых функций:

$$f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x}), \cdots f_n(\mathbf{x}),$$

где *x* - возможное решение. Если новое возможное решение *y* окажется лучше решения *x*, т.е.,

$$f_i(\mathbf{y}) > f_i(\mathbf{x}) \text{ for } \forall i$$

мы можем сказать

“*y* доминирует над *x*”.

Если же такое *y* отсутствует, мы скажем

“x недоминируем” или “Паретто-оптимален”

Простейший пример: Допустим, у нас имеются две целевые функции.

$$\begin{cases} f_1(x) = x^2 \\ f_2(x) = (x-2)^2 \end{cases}$$

- $x=0$ – является оптимальным решением функции f_1, но не для f_2.
- $x=2$ – является оптимальным решением функции f_2, но не для f_1.
- Любое значение между двумя указанными будет компромиссным решением или Паретто-оптимальным.
- Однако решение $x=3$ не является Паретто-оптимальным, поскольку эта точка не лучше решения $x=2$ для заданных целевых функций.
- Если мы построим график в пространстве решений f1-f2, то увеличение f1 на некотором интервале повлечет уменьшение f2, и, наоборот, что означает, что решения в этом интервале будут Паретто-оптимальными. В другом же интервале значений функции f1 увеличение f1 приведет к росту значения функции f2 (и наоборот). См. Рисунок 32. Пространство f1-f2 еще называется *Пространством Компромиссного Решения* (Trade-off Space).

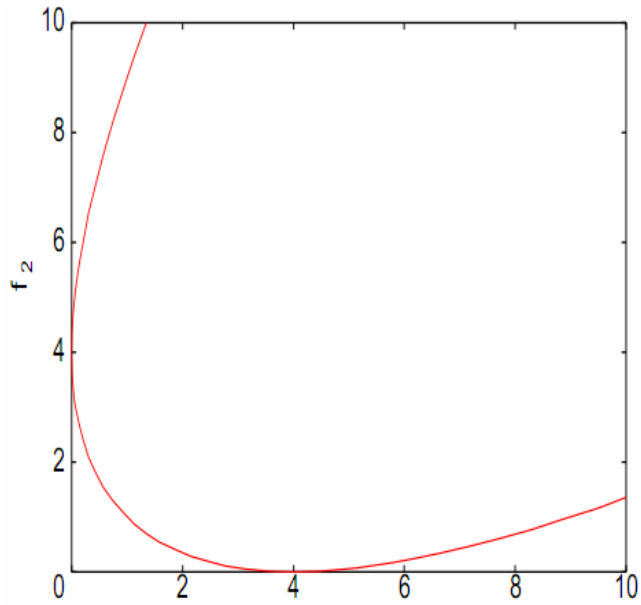


Рисунок 32: Пространство Компромиссного Решения функций $f_1(x) = x^2$ и $f_2(x) = (x-2)^2$.

Тема для размышления: Интересно попробовать графически отобразить все особи популяции 0 и, допустим, популяции 100?

Как использовать несколько целевых функций? Эволюционный процесс в этом случае в чем-то похож на применение генетического алгоритма с одной функцией приспособленности. Единственное отличие в том, что мы имеем несколько целевых функций. Далее мы объединяем множество целевых функций в единую функцию приспособленности. Существует множество способов решения этой задачи. Перечислим некоторые их них:

- Использование взвешенной суммы.

Функция приспособленности в этом случае рассчитывается по формуле

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i f_i(\mathbf{x}) \tag{16}$$

где w_i характеризует важность i-й целевой функции.

Обратите внимание на то, что для любого множества весов > 0, оптимальное решение “недоминируемое”, однако обратное утверждение не всегда верно.

- Минимаксный подход

  Функция приспособленности рассчитывается как результат минимизации максимума n целевых функций.

- Использование вектора целей

  Функция приспособленности рассчитывается посредством минимизации вектора

  $$(f_1, f_2, f_3, \cdots, f_n)$$

  от заранее заданной цели.

- Подход усредненного ранжирования

  В данном случае рассчитывается ранг $r(x_i)$ для каждой особи x_i в популяции с учетом i-ой целевой функции. Оценка приспособленности определяется как усредненное значение всех r_i ($i=1,\ldots,n$).

- Подход ранжирования Паретто.

  Ранжирование выполняется согласно “количеству особей в популяции, которые над ними доминируют”.

Рассмотрим общий алгоритм применения многоцелевого генетического алгоритма.

1. *Генерирование популяции.*
2. *Выбор особей в популяции.*
3. *Выполнение скрещивания и мутации для генерации потомка.*
4. *Расчет ранга для полученного потомка.*
5. *Поиск особи в популяции максимально похожей на сформированного потомка. Замещение этой особи потомком, в случае если ранг потомка выше или если потомок доминирует* [5].
6. *Если потомок был принят, то пересчет рангов в популяции.*
7. *Повтор шагов 2-6 в соответствии с размером популяции.*

[5] На 5-ом шаге алгоритма добавление нового потомка в популяцию происходит только при условии, что он доминирует над наиболее схожей с ним особью, или если она имеет более низкий ранг, т.е. ниже степень доминирования.
Такая стратегия привносит элементы элитизма в вычисления, поскольку недоминируемая особь может быть замещена только потомком, который доминирует над ней.
Степень подобия двух особей рассчитывается через функцию расстояния.

8. *Если желаемый результат не был достигнут, вернуться к шагу 2 и сформировать новую популяцию.*

16.1 Возвращаясь к управлению роботом
- Исследования любителей марса

17 Эволюция структуры
- **от простых структур к сложным**

17.1 Конечные автоматы

Рисунок 33: Пространство Компромиссного Решения
функций $f_1(x) = x^2$ and $f_2(x) = (x-2)^2$.

18 Беспорядочный Генетический Алгоритм (Messy Genetic Algorithm – m-GA)

Таким образом, до сих пор мы использовали хромосомы фиксированной длины. Конечно же, удобно работать с хромосомами фиксированной длинны, но иногда требуется большая гибкость. Поэтому в этом разделе рассмотрим, как можно применить хромосомы произвольной длинны. Допустим, имеется следующая хромосома:

(1, 0, 0, 1, 1)

В m-GA эта хромосома представляется в виде

((1,1), (2,0), (3,0), (4,1), (5,1))

18.1 Теперь позиция каждого отдельного гена не имеет значения

Как вы уже, наверное, догадались, каждый ген представляется парой чисел (1) его местоположением и (2) его значением. Например, ген (1,1) является *1-ым геном хромосомы и принимает значение 1*. Аналогично для гена (2,0) – является *2-ым геном хромосомы и принимает значение 0*. Теперь, когда у нас каждый ген несет информацию о своей позиции в хромосоме, порядок генов в m-GA соблюдать не обязательно. Т.е. хромосомы

((2,0), (4,1), (1,1), (5,1), (3,0)) и ((4,1), (1,1), (5,1), (2,0), (3,0))

одинаковы.

18.2 Разделяй и соединяй

Операция скрещивания, когда мы имеем дело с m-GA, характеризуется возможностью выполнять разделение родительских хромосом в любой позиции, в то время, как рассмотренный нами одноточечный алгоритм скрещивания, предполагал деление родительских хромосом в одной и той же позиции. Далее происходит обмен между родительскими парами участками хромосом и получение потомков. Продемонстрируем на примере. Пусть имеется две родительские хромосомы

((4,1), (1,1), (5,1), (2,0), (3,0)) и ((3,0), (5,0), (4,1), (1,0), (2,0))

в случае m-GA мы можем разделить первую родительскую особь в позиции между 1-ым и 2-ым геном, а вторую особь между 4-ым и 5-ым генами. В результате образуются две дочерние особи следующего вида

((4,1), (2,0)) и ((3,0), (5,0), (4,1), (1,0), (1,1), (5,1), (2,0), (3,0))

18.3 Ситуация Пере-определенности (Over-specification) и Недо-определенности (Under-specification)

В рассмотренном нами примере после выполнения операции скрещивания некоторые позиции генов в дочерних хромосомах могут оказаться включенными несколько раз или отсутствовать вообще. Первая ситуация получила название *Пере-определенность*, а вторая - *Недо-определенность*. Обычно *Пере-определенность* разрешается по принципу “первым вошел первым получил обслуживание” (как в ресторанах McDonald).
Т.е.:

((3,0), (5,0), (4,1), (1,0), (1,1), (5,1), (2,0), (3,0))

рассматривается как

((3,0), (5,0), (4,1), (1,0), (2,0)).

Если эту запись привести к виду обычного генетического алгоритма, то получим хромосому

(0 0 0 1 0).

В случае *Недо-определенности* один из возможных вариантов – дополнить недостающие гены, сгенерировав их случайным образом. Иногда можно просто не обращать внимание на отсутствующие гены (это зависит от специфики решаемой задачи).

18.4 Примеры

18.4.1 База Правил

18.4.2 Задача о рюкзаке – пересмотр

18.4.3 В чем же заключается преимущество?

18.5 Создание нейросетевых структур при помощи m-GA

Допустим каждый из наших генов включает в свой состав четыре подгена

(…, ((6), (8, 0.28, 1)), ((7), (12, -0.73, 0)), …).

Далее, предположим, что два гена из хромосомы, приведенной выше, интерпретируются следующим образом:

- 6-ой ген должен быть соединен с 8-ым геном синоптической связью весом 0,28, и данное соединение активно.
- 7-ой ген должен быть соединен с 12-ым геном синоптической связью весом – -0,73, но соединение неактивно.

Другими словами, *n*-ый ген задает (1) с каким нейроном соединен *n*-ый нейрон; (2) весовой коэффициент соединения; (3) активно соединение или нет.

19 EP и ES – модификации GP, но гораздо старше

20 Генетическое программирование (Genetic Programming - GP)

В названии данного раздела под *Генетическим Программированием* понимается *Эволюция Программ*. Это означает использование эволюционных вычислений при подготовки кода программы для решения некоторой задачи. Начальную популяцию составляют случайные коды программ, которые впоследствии вовлекаются в эволюционный процесс. Предполагается, что каждая последующая популяция будет содержать коды более совершенных программ, чем предыдущая.

Как и ранее, (i) создается популяция случайных хромосом; (ii) оценивается приспособленность каждой хромосомы; (iii) выбираются хромосомы таким образом, что вероятность образовать пару выше у хромосом, относящихся к лучшим представителям популяции; (iv) в результате скрещивания и мутации получают потомков; (v) повторяются пункты с (iii) по (iv) пока в новой популяции количество хромосом не достигнет их количества в предыдущей популяции; (vi) повторяются пункты с (ii) по (v), до тех пор, пока качество решения не достигнет необходимого уровня или изменения в популяции прекратятся.

Но в данном случае отдельная хромосома представляется не строкой, а деревом.

20.1 Как случайным образом сгенерировать дерево, и как выполнить эволюцию деревьев

У нас имеется множество функций и множество завершающих элементов. Далее мы следуем пунктам алгоритма, приведенного ниже:

1) Случайным образом из *множества функций* выбирается та функция, которая будет выполнять роль корневого узла дерева.
2) Для каждого ребра, опять же, случайным образом выбирается своя функция или завершающий элемент из соответствующих множ*еств функций и завершающих элементов*.
3) Если узел является завершающим, то дальнейшее развитие этой ветви дерева прекращается. Узел становится *листом* дерева. Иначе повторяется пункт (2).
4) Алгоритм повторяется до тех пор, пока все оконечные узлы не станут завершающими.

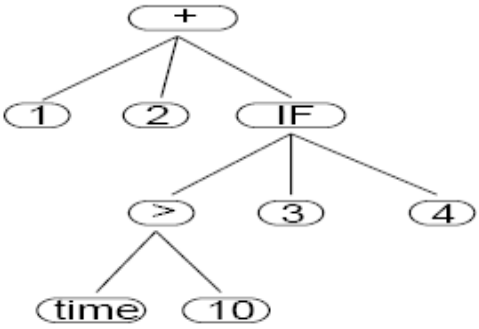
20.2 Эволюция программ

Некоторые языки программирования имеют структуру в виде дерева (например, язык программирования *LISP*).

20.2.1 Пример записи программы в виде дерева

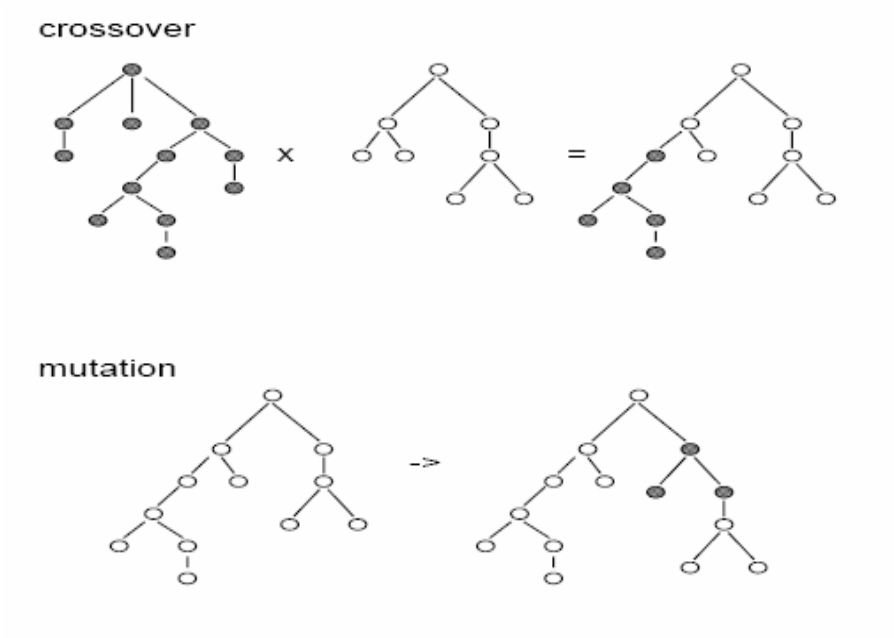
Программа на языке LISP может быть представлена в виде дерева. Здесь приведен простейший пример одной инструкции на LISP, и дерева, задающего ее структуру.

```
(+ 1 2 (IF (> time 10) 3 4))
```



20.2.2 Кроссовер и мутация

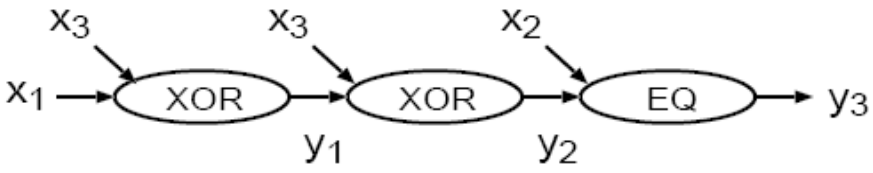
Рассмотрим как выполняется операция кроссовера и мутации двух деревьев (см. рисунки ниже).

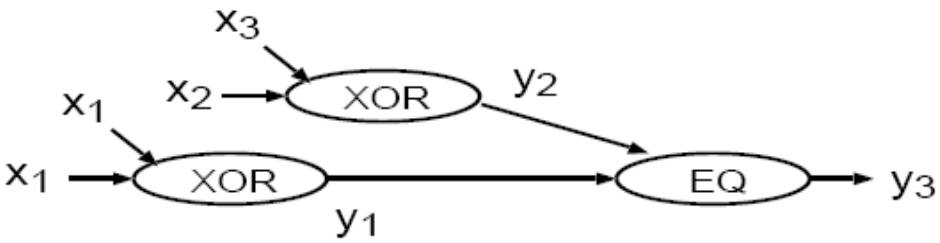


20.3 Эволюция аппаратных средств

20.3.1 Even-n-parity

Иногда нам требуется возможность автоматического обнаружения ошибок при кодировании. Предположим, что необходимо закодировать (n-1) бинарных бита. Для того чтобы обнаружить случайное изменение кода, мы добавляет дополнительный бит, который еще называют *битом контроля равенства* (parity-check-bit). Значение, присваиваемое этому биту (0 или 1), определяется по четности или нечетности количества 1. Таким образом, появляется возможность обнаружить неправильно закодированную последовательность. Хотя отдельные ошибки могут быть пропущены.





21 Ландшафт приспособленности (Fitness Surface)

На Рисунке 30 схематически изображен ландшафт приспособленности в некотором двумерном пространстве. Для каждой возможной точки из области поиска рассчитано и отображено соответствующее значение приспособленности. Если пространство некоторой прикладной области будет более чем трехмерное, то мы не способны выполнить визуализацию, хотя возможно мысленно представить себе гиперповерхность. Эта (гипер-)поверхность называется еще ландшафтом приспособленности, и характеризуется наличием пиков, при чем вершина самого высокого пика соответствует глобальному решению, а вершины остальных пиков – локальным оптимумам.

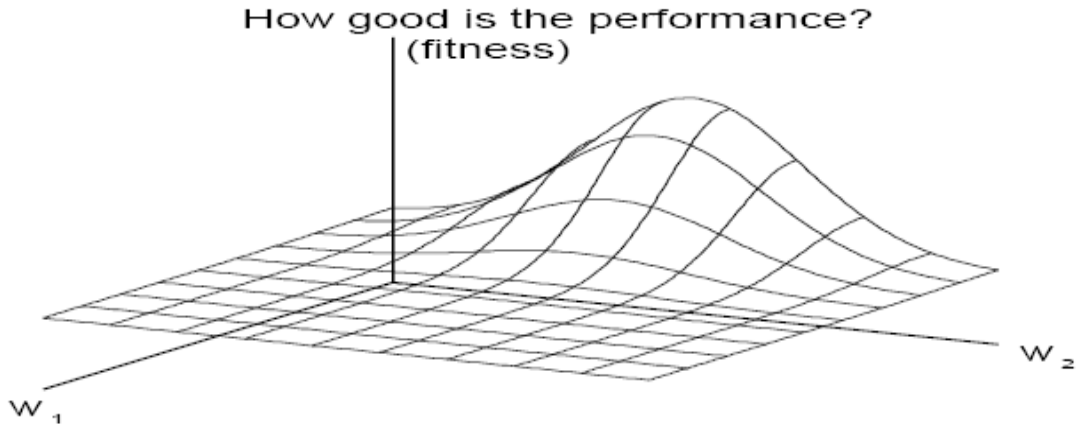


Рисунок 30: Схематическое изображение приспособленности в некотором двумерном пространстве.

21.1 Движение к вершине холма

-- к вершине холма методом случайных мутаций --
(Random Mutation Hill-climbing (RMHC))

(1) Сгенерируйте случайным образом последовательность и считайте ее текущей вершиной холма.
(2) Случайным образом определите точку в последовательности для инвертирования. Если в результате инвертирования значение приспособленности не изменится или возрастет, то текущей вершиной холма становится новая последовательность.
(3) Повторяйте пункт (2) до тех пор, пока не будет получена оптимальная последовательность или не будет превышено допустимое количество повторов.
(4) Возвратитесь к текущей вершине холма.

-- метод крутого подъема к вершине холма --
(Steepest Ascent Hill-climbing (SAHC))

(1) Сгенерируйте случайным образом последовательность и считайте ее текущей вершиной холма.
(2) В направлении слева направо инвертируйте каждый бит, запоминая значение приспособленности.

(3) Если в результате какая-либо последовательность увеличивает приспособленность, то сделайте текущей вершиной холма эту последовательность (связи выбираются случайным образом).
(4) Если увеличения приспособленности не происходит, то сохраните текущую вершину и возвращайтесь к (1), в ином случае – к (2) но уже с новой вершиной холма.

-- Метод следующего подъема к вершине холма --
(Next Ascent Hill-climbing (NAHC))

(1) Сгенерируйте случайным образом последовательность и считайте ее текущей вершиной холма.
(2) То же что и в SAHC, за исключением отдельных моментов: как только произошло увеличение приспособленности, возвращайтесь к началу пункта (2) с новым значением текущей вершины, не инвертируя оставшиеся биты. После возвращения начните с бита в позиции, где было обнаружено увеличение значения приспособленности.
(3) Если увеличения не происходит, то возвращайтесь к пункту (1).

21.2 Использование популяция таких последовательностей

21.2.1 Эволюционное программирование (Evolutionary Programming - EP)

21.2.2 Эволюционная стратегия (Evolution Strategy - ES) – цель - повысить эффективность

21.3 Особые ландшафты

Как быть в случае, если ландшафт не имеет плавных переходов?

21.4 even-n-parity – пересмотр

22 Иголка в стогу сена

Сегодня мы все чаще используем банкоматы, с помощью которых мы можем обращаться к собственному счету в банке. Для доступа к счету используется PIN-код, состоящий обычно из четырех десятичных чисел. В целях безопасности сделано так, что если мы три раза подряд неправильно вводим PIN, то он становится недействительным. Нас интересует вопрос: “Сколько попыток потребуется для того, чтобы подобрать код, если количество попыток не ограничено?”. Формализуем эту проблему.

22.0.1 Взлом кода PIN

Предположим, что для построения PIN используется p-битные восьмеричные числа[6], и только одна из 8^p возможных комбинаций является секретным PIN кодом. Никому кроме владельца PIN код не известен. Поэтому задача следующая: “Сколько попыток понадобится, чтобы подобрать код, придерживаясь определенной тактики (стратегии)?”

Это напоминает известную задачу, называемую иголка в стоге сена, которая была предложена Хинтоном и Новланом (Hinton & Nowlan) в 1987 [?]. Иголка в этой задаче представлялась одной из возможных комбинаций 20 бит в строке, а пространство поиска включало 2^{20} точек. Т.е. только одна точка в этом пространстве является иглой, которую необходимо найти. Любая дополнительной информации о том, как близко выбранная в данный момент точка располагается к игле, или насколько выбранная точка похожа на иглу, отсутствует.

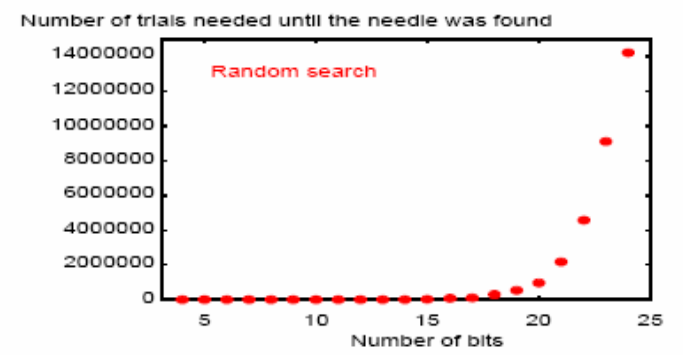


Рисунок 31: При увеличении количества бит поиск усложняется экспоненциально.

22.1 Маленький остров в большом озере

Ситуация, когда пик представляется в виде горы с плоской вершиной и особенно крутыми склонами, а вокруг равнина – т.н. нулевой уровень, получила название “Маленький остров в большом озере”. А особенно затруднительная ситуация, когда только одна точка из пространства поиска соответствует уровню один, а все остальные точки имеют нулевой уровень – “Иголка в стоге сена” (см. Рисунок).

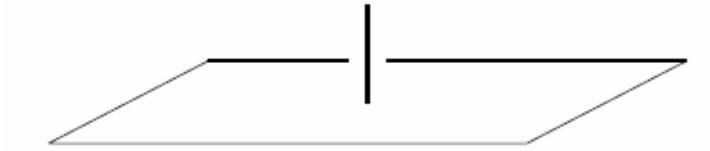


Рисунок 32: Иголка в стоге сена.

[6] Вы поймете, почему используются восьмеричные числа вместо десятичных позже, когда в разделе “Эксперименты” ознакомитесь с подразделом посвященным “интрону”

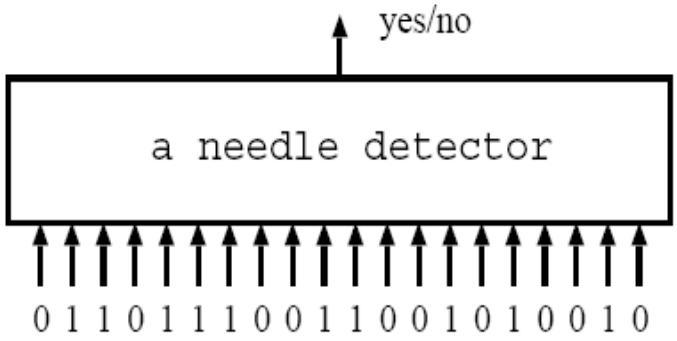
23 Почему у жирафа длинная шея?
- Исследование Ламарка и Эффект Болдуина

23.1 Иголка в стоге сена – самая сложная задача

Однажды Хинтоном и Новланом был проведен очень интересный эксперимент по поиску иглы в стоге сена. Их игла и стог сена были заданы следующим образом.

- Игла –> одна из возможных конфигураций 20-битной строки.
- Стог сена –> (2^{20}-1) точек для выполнения поиска.

Например, следующая последовательность (11111111110000000000) соответствует единице, а все остальные – нулю. Мы можем представить, что задан черный ящик, как изображено на рисунке, и наша задача



заключается в подборе 20 бит на входе ящика, таким образом, что бы на выходе было возвращено значение “YES”. Только одна 20-битная последовательность из имеющихся 2^{20} битовых последовательностей является решением.

Хинтон и Новлан предложили использовать обучение каждой особи в течение ее жизни. Если результат обучения влияет на следующее поколение, то такая ситуация называется эффектом Болдуина (как в биологической эволюции). У Хинтона и Новлана это выглядит следующим образом:

- обучение каждой особи в течении жизни (эффект Болдуина)
  - около 25% принимают значение равное “1”, еще 25% - значение “0”, а оставшиеся 50% - “?”.
  - Эволюция происходит в результате подстановки случайным образом выбираемых “1” или “0” в позиции, отмеченные символом “?” -> обучение.
  - Для каждой особи обучение повторяется до 1000 раз -> обучение в течение жизни.
  - Если достигнута точка приспособленности равная единице за *n* попыток, то уровень, которого достигло обучение, рассчитывается по формуле

$$1+19*(1000-n)/1000$$

24 Нечеткая Логика

24.1 Нечеткий Контроллер

Постановка нашей задачи

Предположим, что *x* – это скорость моего автомобиля, *y* – расстояние до автомобиля спереди, а *z* – сила, с которой мы давим на педаль тормоза. Обеспечим управление моим автомобилем при помощи следующего набора правил:

- IF *x* есть *быстро* и *y* THEN z должно быть *сильно*
- IF *x* есть *средне* и *y* есть *далеко* THEN z должно быть *средне*
- IF *x* есть *довольно медленно* или *x* есть *средне* и *y* есть *далеко* THEN *z* должно быть *слабо*
- IF *x* есть *медленно* или *x* есть *довольно медленно* и *y* есть *близко* или *y* есть *довольно близко* THEN z должно быть *довольно слабо*
- и т.д.

В таком случае результат может быть представлен как на Рисунке ниже.

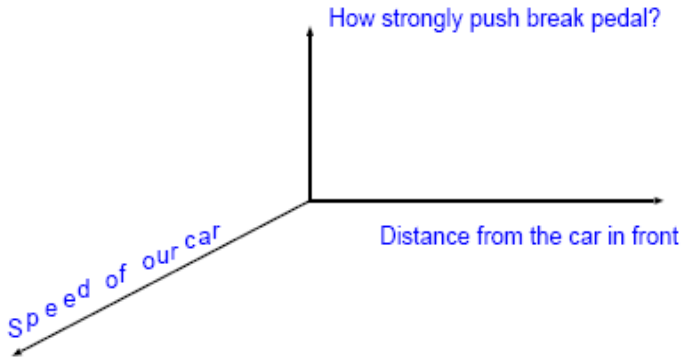


Рисунок 34: Образец использования нечеткого контроллера.

Например, как должны мы поступить, когда *x*=80 км/ч, а *y*=40 м?

Нечеткое множество и классическое (четкое) множество

- $0 < x < 10$
- $x=12$
- {*x* гораздо меньше 10}
- {*x* около 12}
- Пиво либо {очень холодное, холодное, не очень холодное, теплое}

Функция принадлежности

В какой степени x соответствует A задается функцией принадлежности $\mu_A(x)$.
{*x* около 12}

$$\mu(x) = \frac{1}{1 + (x - 12)^2} \tag{17}$$

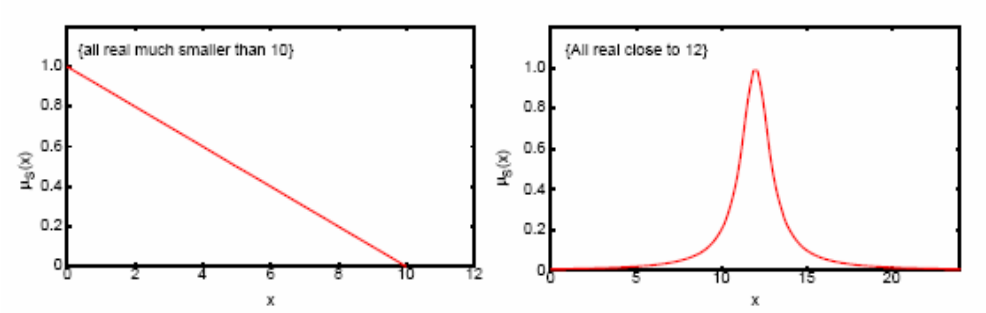


Рисунок 35: Примеры функций принадлежности {*x* гораздо меньше 10} (слева) и {*x* около 12} (справа).

AND и OR в нечеткой логике

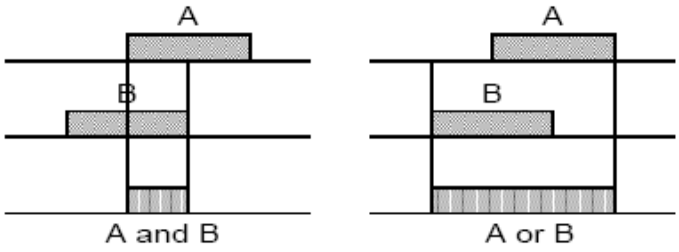


Рисунок 36: AND и OR в четкой логике.

В нечеткой логике, функции принадлежности *AandB* и *AorB* можно определять по-разному, но наиболее часто используются

$$\mu_{AandB}(x) = \min\{\mu_A(x), \mu_B(x)\} \tag{18}$$

и

$$\mu_{AorB}(x) = \max\{\mu_A(x), \mu_B(x)\}, \tag{19}$$

соответственно.

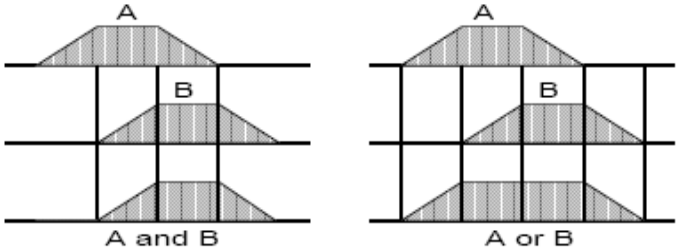


Рисунок 37: AND и OR в нечеткой логике.

Следующий вопрос, каким образом можно визуализировать функцию принадлежности, в случае, когда функция определена более одного раза (как в примерах выше)?

- Очень холодное и холодное пиво.

(μ задает температуру)
Давайте попробуем…
Выглядит необычно, не правда ли? Также существует большое количество других определений, например, определение Лукашевича (Lucasiewics)

$$\mu_{AandB}(x) = \max\{1, \mu_A(x) + \mu_B(x)\} \quad (20)$$

и

$$\mu_{AorB}(x) = \min\{1, \mu_A(x) + \mu_B(x)\}, \quad (21)$$

соответственно.

- Молодой и высокий.

1. 3-D графика (*z*= μ задается для *x*=”возраст” и *y*=”рост”)
2. Представление в матричном виде.

Использование правил IF-THEN в нечеткой логике

В нечеткой логике, принадлежность *IF A Then B* также можно задавать несколькими способами. Рассмотрим следующие варианты:

Метод Mamdani

$$\mu_{A\to B}(x) = \min\{\mu_A(x), \mu_B(x)\} \quad (22)$$

Метод Larsen

$$\mu_{A\to B}(x) = \mu_A(x) \times \mu_B(x) \quad (23)$$

24.1.1 Возвращаясь к управлению автомобилем

Пример 1
IF *x=быстро* THEN *z =сильно*

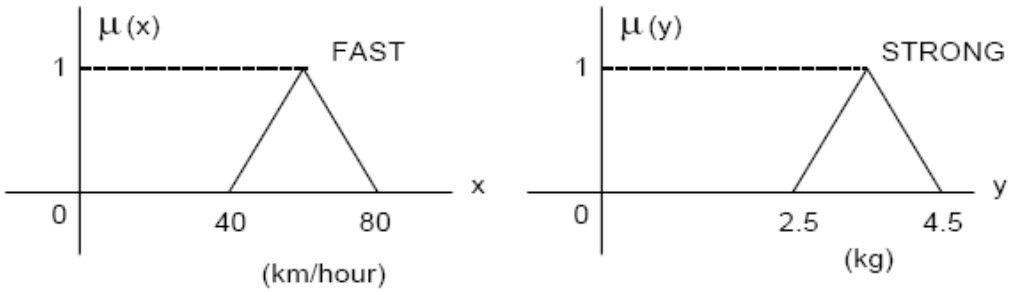


Рисунок 38: Примеры функций принадлежности {*x* гораздо меньше 10} (слева) и {*x* около 12} (справа).

x \ y	2.5	3.0	3.5	4.0	4.5
40	0.0	0.0	0.0	0.0	0.0
50	0.0	0.5	0.5	0.5	0.0
60	0.0	0.5	1.0	0.5	0.0
70	0.0	0.5	0.5	0.5	0.0
80	0.0	0.0	0.0	0.0	0.0

Рисунок 39: Примеры функций принадлежности {*x* гораздо меньше 10} (слева) и {*x* около 12} (справа).

Пример 2 … два правила с 1 элементом
R_1: IF *x=медленно* THEN *z=быстро*
R_2: IF *x=средне* THEN *z=средне*

Пример 2 … 1 правило 2 элемента
R_1: IF *x=медленно* AND *y=средне* THEN *z=сильно*

Никакой более графики или матриц, но…
Выходное нечеткое множество, когда **x**=40 выглядит как
Это не правильно

24.2 Нечеткость и генетический алгоритм

эволюция функции принадлежности

- Треуголная функция принадлежности
- Гауссовская

Хромосома, которая соответствует привилу.

25 Коллективное интеллект

25.1 Оптимизация деятельности колонии муравьев

25.1.1 Использование Поиска данных (Data-mining)

Setosa				Versicolor				Virginica			
x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
0.65	0.80	0.20	0.08	0.89	0.73	0.68	0.56	0.80	0.75	0.87	1.00
0.62	0.68	0.20	0.08	0.81	0.73	0.65	0.60	0.73	0.61	0.74	0.76
0.59	0.73	0.19	0.08	0.87	0.70	0.71	0.60	0.90	0.68	0.86	0.84
0.58	0.70	0.22	0.08	0.70	0.52	0.58	0.52	0.80	0.66	0.81	0.72
0.63	0.82	0.20	0.08	0.82	0.64	0.67	0.60	0.82	0.68	0.84	0.88
0.68	0.89	0.25	0.16	0.72	0.64	0.65	0.52	0.96	0.68	0.96	0.84
0.58	0.77	0.20	0.12	0.80	0.75	0.68	0.64	0.62	0.57	0.65	0.68
0.63	0.77	0.22	0.08	0.62	0.55	0.48	0.40	0.92	0.66	0.91	0.72
0.56	0.66	0.20	0.08	0.84	0.66	0.67	0.52	0.85	0.57	0.84	0.72
0.62	0.70	0.22	0.04	0.66	0.61	0.57	0.56	0.91	0.82	0.88	1.00
0.68	0.84	0.22	0.08	0.63	0.45	0.51	0.40	0.82	0.73	0.74	0.80
0.61	0.77	0.23	0.08	0.75	0.68	0.61	0.60	0.81	0.61	0.77	0.76
0.61	0.68	0.20	0.04	0.76	0.50	0.58	0.40	0.86	0.68	0.80	0.84
0.54	0.68	0.16	0.04	0.77	0.66	0.68	0.56	0.72	0.57	0.72	0.80
0.73	0.91	0.17	0.08	0.71	0.66	0.52	0.52	0.73	0.64	0.74	0.96
0.72	1.00	0.22	0.16	0.85	0.70	0.64	0.56	0.81	0.73	0.77	0.92
0.68	0.89	0.19	0.16	0.71	0.68	0.65	0.60	0.82	0.68	0.80	0.72
0.65	0.80	0.20	0.12	0.73	0.61	0.59	0.40	0.97	0.86	0.97	0.88
0.72	0.86	0.25	0.12	0.78	0.50	0.65	0.60	0.97	0.59	1.00	0.92
0.65	0.86	0.22	0.12	0.71	0.57	0.57	0.44	0.76	0.50	0.72	0.60
0.68	0.77	0.25	0.08	0.75	0.73	0.70	0.72	0.87	0.73	0.83	0.92
0.65	0.84	0.22	0.16	0.77	0.64	0.58	0.52	0.71	0.64	0.71	0.80
0.58	0.82	0.14	0.08	0.80	0.57	0.71	0.60	0.97	0.64	0.97	0.80
0.65	0.75	0.25	0.20	0.77	0.64	0.68	0.48	0.80	0.61	0.71	0.72
0.61	0.77	0.28	0.08	0.81	0.66	0.62	0.52	0.85	0.75	0.83	0.84
0.63	0.68	0.23	0.08	0.84	0.68	0.64	0.56	0.91	0.73	0.87	0.72
0.63	0.77	0.23	0.16	0.86	0.64	0.70	0.56	0.78	0.64	0.70	0.72
0.66	0.80	0.22	0.08	0.85	0.68	0.72	0.68	0.77	0.68	0.71	0.72
0.66	0.77	0.20	0.08	0.76	0.66	0.65	0.60	0.81	0.64	0.81	0.84
0.59	0.73	0.23	0.08	0.72	0.59	0.51	0.40	0.91	0.68	0.84	0.64
0.61	0.70	0.23	0.08	0.70	0.55	0.55	0.44	0.94	0.64	0.88	0.76
0.68	0.77	0.22	0.16	0.70	0.55	0.54	0.40	1.00	0.86	0.93	0.80
0.66	0.93	0.22	0.04	0.73	0.61	0.57	0.48	0.81	0.64	0.81	0.88
0.70	0.95	0.20	0.08	0.76	0.61	0.74	0.64	0.80	0.64	0.74	0.60
0.62	0.70	0.22	0.04	0.68	0.68	0.65	0.60	0.77	0.59	0.81	0.56
0.63	0.73	0.17	0.08	0.76	0.77	0.65	0.64	0.97	0.68	0.88	0.92
0.70	0.80	0.19	0.08	0.85	0.70	0.68	0.60	0.80	0.77	0.81	0.96

(продолжение на следующей странице)

(продолжение)

Setosa				Versicolor				Virginica			
x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
0.62	0.70	0.22	0.04	0.80	0.52	0.64	0.52	0.81	0.70	0.80	0.72
0.56	0.68	0.19	0.08	0.71	0.68	0.59	0.52	0.76	0.68	0.70	0.72
0.65	0.77	0.22	0.08	0.70	0.57	0.58	0.52	0.87	0.70	0.78	0.84
0.63	0.80	0.19	0.12	0.70	0.59	0.64	0.48	0.85	0.70	0.81	0.96
0.57	0.52	0.19	0.12	0.77	0.68	0.67	0.56	0.87	0.70	0.74	0.92
0.56	0.73	0.19	0.08	0.73	0.59	0.58	0.48	0.73	0.61	0.74	0.76
0.63	0.80	0.23	0.24	0.63	0.52	0.48	0.40	0.86	0.73	0.86	0.92
0.65	0.86	0.28	0.16	0.71	0.61	0.61	0.52	0.85	0.75	0.83	1.00
0.61	0.68	0.20	0.12	0.72	0.68	0.61	0.48	0.85	0.68	0.75	0.92
0.65	0.86	0.23	0.08	0.72	0.66	0.61	0.52	0.80	0.57	0.72	0.76
0.58	0.73	0.20	0.08	0.78	0.66	0.62	0.52	0.82	0.68	0.75	0.80
0.67	0.84	0.22	0.08	0.65	0.57	0.43	0.44	0.78	0.77	0.78	0.92
0.63	0.75	0.20	0.08	0.72	0.64	0.59	0.52	0.75	0.68	0.74	0.72

25.2 Задача распределения заданий (Job Shop Scheduling Problem - JSSP)

Допустим имеется m машин $M_1, \cdots, M_m$ и n заданий $J_1, \cdots, J_n$. Каждое задание J_i состоит из n_j элементарных операций O_{ij} ($i = 1, \cdots n_j$), которые должны выполняться в определенном порядке $O_{1j}, O_{2j}, \cdots O_{nj}$. Каждая операция O_{ij} должна выполниться без приоритетов на выделенной машине $\mu_{ij} \in M_1, \cdots M_m$ в течение определенного промежутка времени p_{ij}.

Каждая задача должна быть выполнена на отдельной машине. Любое задание выполняется на каждой машине в точности один раз. Кроме того, заранее задан определенный порядок выполнения задач в рамках одного задания. В определенный момент времени машина может обрабатывать только одно задание. Нет ни предустановленных сроков, ни временных промежутков. Под временем обработки понимается промежуток от начала выполнения первой задачи и до момента окончания выполнения последней задачи. Цель – определить моменты времени, в которые необходимо запустить на выполнению каждую задачу, при чем таким образом, что бы время обработки было минимальным.

Каждая операция O_{jr} требует эксклюзивного использования ресурсов M_r в течение непрерывного промежутка времени p_{jr}.

25.2.1 Пример

25.3 Искусственный улей

Поведение пчелы, возвратившейся в улей с нектаром, собранным на цветке:

- Если пчела видит, что не следует больше возвращаться к цветку, то нет смысла выполнять танец, лучше обратить внимание на других танцующих пчел, выбрать и проследовать за ними.

- Продолжить использование цветка самостоятельно, не исполняя танец и, соответственно, не привлекая других пчел.

- Исполнять танец и привлекать к найденному цветку других пчел.

Танец характеризуется колебательными движениями, выполняемыми пчелой. Он информирует других пчел о (i) полезности цветка; (ii) расстоянии до цветка; (iii) направлении.

26 Обучение с подкреплением (Reinforcement Learning)

Обучение с подкреплением – это подход из области машинного обучения для решения целевых задач, в которых испытуемая система (агент) принимает решение посредством выбора одного из заранее заданных действий в зависимости от той ситуации, в которой агент оказывается в предшествующий и последующий моменты.
Проще говоря, хотя и не так просто как хотелось бы, обучение с подкреплением – это то, как агент обучается добиваться цели в результате последовательности случайных решений, т.е. циклически происходит изменение ситуации и реагирование агента на нее.

Состояние, действие, стратегия и вознаграждение/наказание

Таким образом, обучение с подкреплением состоит из следующих компонентов (i) *ситуация* – чаще упоминается как *состояние*, т.е. состояние окружающей среды; (ii) *действие*; (iii) *стратегия* – отображение ситуации в действие; (iv) *вознаграждение/наказание*, которые присуждаются агенту в зависимости от выбранного им действия в конкретной ситуации.

В качестве примера будут приведены задачи “автомобиля на холме”, “джипа” и т.п.

Оценка состояния

Каждому состоянию сопоставлено значение, которое называется оценкой состояния. Это значение представляет собой сумму подкреплений, полученных в ходе выполнения заданной стратегии при запуске из данного состояния и до завершающего состояния.

Функция оценки

Функция оценки – это отображение из состояний в оценки состояний.

Оптимальная стратегия

Поэтому оптимальной стратегией будет такое отображение из состояний в действия, которое максимизирует суммарное значение подкреплений при запуске из произвольного состояния и до тех пор, пока не будет достигнуто конечное состояние.

Цель агента – определить такую стратегию, которая максимизировала бы полученное им вознаграждение за определенный промежуток времени. Чтобы получить такую стратегию, нужно задать величину *ожидаемого вознаграждения* при запуске из состояния *s*, выборе действия *a* и переходе к π,

$$Q^{\pi}(s,a) = E_{\pi}\langle r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \cdots | s_t = s, a_t = a \rangle. \tag{24}$$

Оптимальное *Q* можно определить из выражения

$$Q^{\star}(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{25}$$

Вопрос заключается в том, как получить оптимальное значение *Q(s,a)*. Рассмотрим два варианта. Первый из них называется *Q-обучение* (Q-learning) и рассчитывается согласно следующему выражению:

$$Q(s_t,a_t) := Q(s_t,a_t) + \alpha\{r_{t+1} + \gamma \cdot \max_{a' \in A} Q(s_{t+1},a') - Q(s_t,a_t)\}. \tag{26}$$

Второй называется *SALSA* и рассчитывается:

$$Q(s_t,a_t) := Q(s_t,a_t) + \alpha\{r_{t+1} + \gamma \cdot Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\} \tag{27}$$

В обоих случаях действие a_t на каждом этапе *t* выбирается случайным образом с вероятностью ε, а действие с наивысшим значением *Q* - с вероятностью 1-ε.