УНИВЕРСИТЕТ
искусственного
интеллекта

# Введение в генетические алгоритмы

# Генетический алгоритм

Эволюционное моделирование (evolutionary computation) – направление в искусственном интеллекте, в основе которого лежат принципы, заимствованные из эволюционной биологии и популяционной генетики и объединяющие компьютерные методы моделирования эволюционных процессов в искусственных системах.

Основной идеей генетических алгоритмов является организация «борьбы за существование» и «естественного отбора» среди этих пробных решений. Запишем пробные решения в двоичной форме:

[0100, 0111, 1101, 1100]

[AABB, AAAB, BBBA, BBBB]

Поскольку генетические алгоритмы используют биологические аналогии, то и терминология, которая применяется, напоминает биологическую. Так одно пробное решение, записанное в типовой (чаще двоичной) форме, мы будем называть ***ботом***, а набор всех пробных решений – ***популяцией***. Как известно, принцип естественного отбора заключается в том, что в конкурентной борьбе выживает наиболее приспособленный.

В нашем случае приспособленность особи определяется **целевой (оценочной) функцией**: чем меньше значение этой функции, тем более приспособленным является бот, т. е. пробное решение, использовавшееся в качестве аргумента целевой функции. Размер популяции устанавливается вручную.

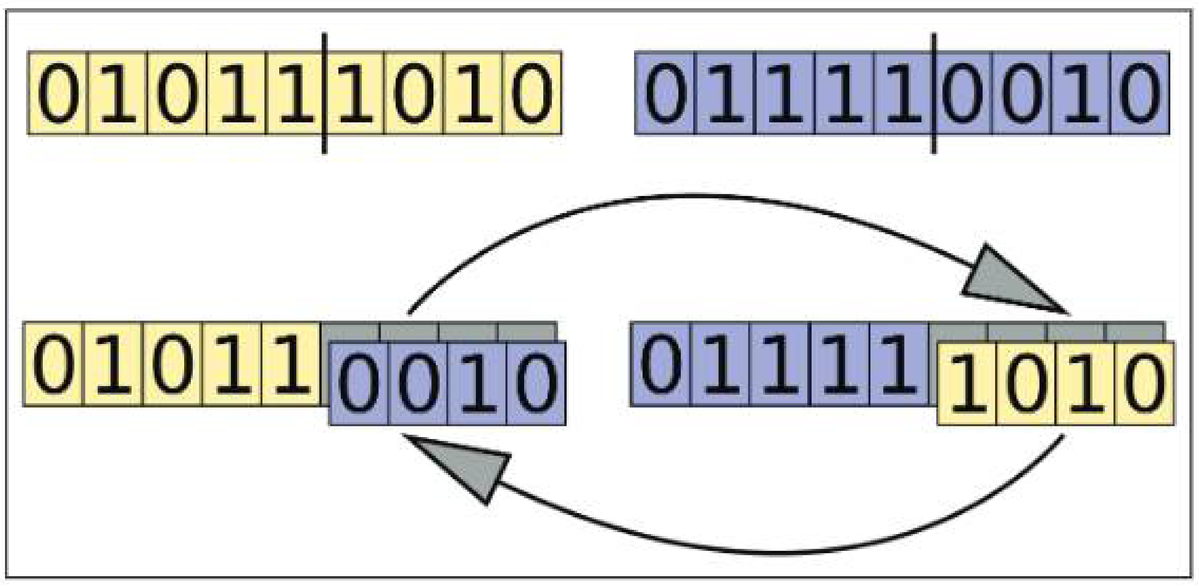
Области применения генетических алгоритмов:

- автоматизации решения оптимизационных задач науки и техники
- изучение и моделирования процессов естественной эволюции
- алгоритмизация перебора гиперпараметров нейросетей
- боты, роботы, агенты (обучение)
- предобработка данных
- планирование

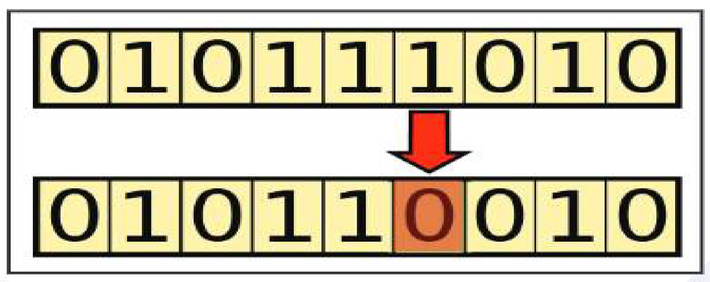
Генетические алгоритмы для нахождения лучшей оценочной функции используют механизм ***естественного отбора***. Они работают с совокупностью ботов – популяцией, каждая из которых представляет возможное решение данной проблемы. Каждая особь оценивается мерой ее «приспособленности» согласно тому, насколько «хорошо» соответствующее ей решение задачи.

# Генетический алгоритм

Наиболее приспособленные боты получают возможность «воспроизводить» потомство с помощью «***перекрестного скрещивания***» с другими особями популяции. Это приводит к появлению новых особей, которые сочетают в себе некоторые характеристики, наследуемые ими от родителей. Наименее приспособленные особи с меньшей вероятностью смогут воспроизвести потомков, так что те свойства, которыми они обладали, будут постепенно исчезать из популяции в процессе эволюции.



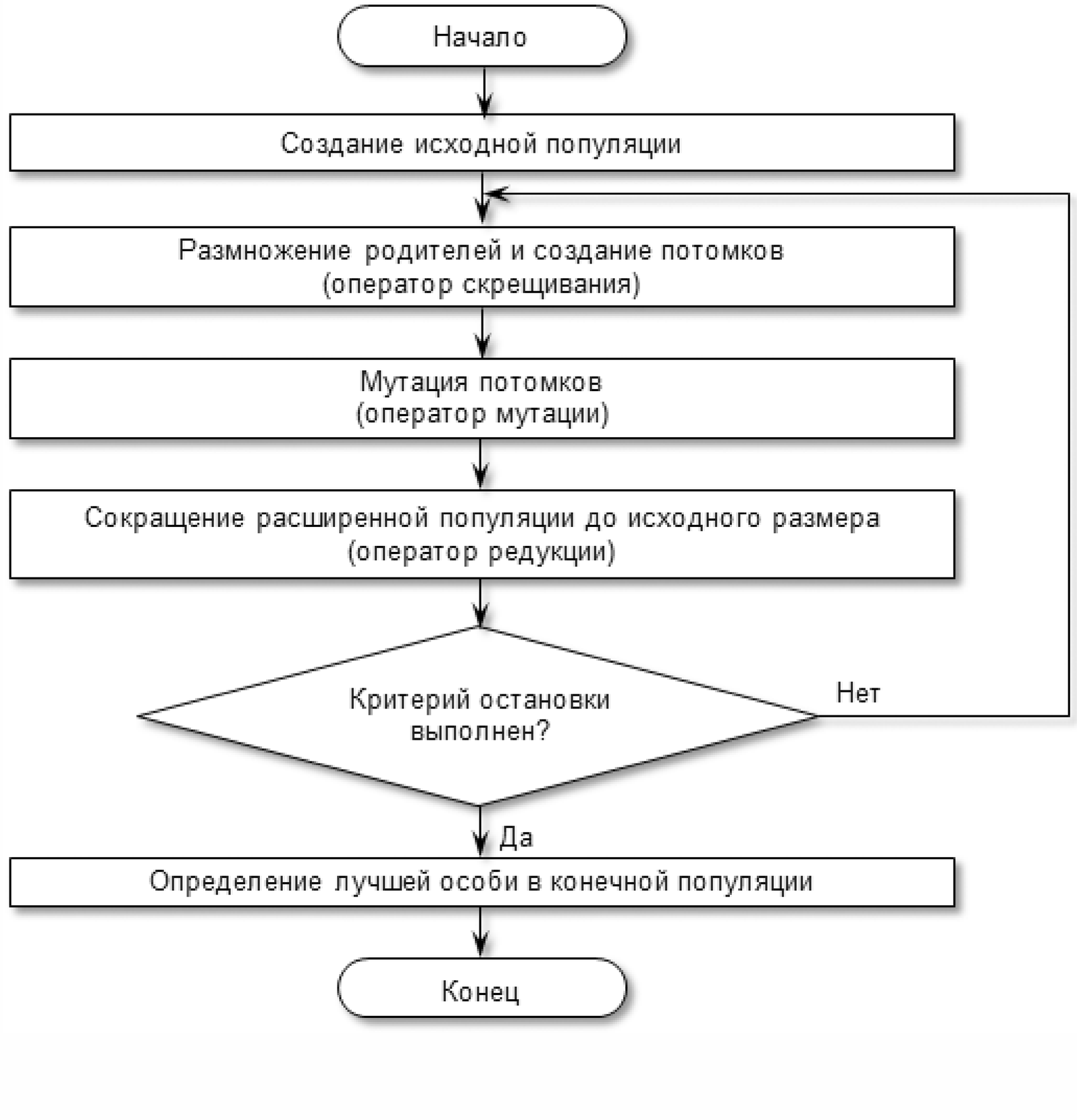

Также в генетическом алгоритме используется метод ***мутации*** – периодически случайным образом обновлять популяцию, т. е. вносить новые сочетания ботов, стимулируя тем самым поиск в неисследованных областях пространства решений. Мутация может проявляться как случайное изменение ботов. Мутации реализуются с помощью внесения случайных изменений в значения ботов с установленным коэффициентом.




Таким образом воспроизводится вся новая популяция допустимых решений, выбирая лучших представителей предыдущего поколения, скрещивая их и получая множество новых особей. Это новое поколение содержит более высокое соотношение характеристик, которыми обладают хорошие члены предыдущего поколения. Таким образом из поколения в поколение хорошие характеристики распространяются по всей популяции. Скрещивание наиболее приспособленных особей приводит к тому, что исследуются наиболее перспективные участки

# Генетический алгоритм

пространства поиска. В конечном итоге **популяция будет сходиться к оптимальному решению задачи.**




Некоторые особенности генетических алгоритмов:

- используют не параметры задачи, а их закодированный вид
- ведут поиск, который исходит не из единственной точки, а из полной популяции
- используют только функцию оценки и никакой вспомогательной информации
- применяют вероятностные, а не детерминированные правила выбора
- в процессе поиска используется значение функции, а не ее приращения

# Генетический алгоритм

Хотя модель эволюционного развития, применяемая в генетических алгоритмах, сильно упрощена по сравнению со своим природным аналогом, тем не менее генетические алгоритмы являются достаточно мощным средством и могут с успехом применяться для широкого класса прикладных задач, включая те, которые трудно, а иногда и вовсе невозможно, решить другими методами. Однако генетические алгоритмы не гарантируют обнаружения глобального решения за конечное время. Генетические алгоритмы не гарантируют и того, что глобальное решение будет найдено (впрочем, для произвольной целевой функции за конечное время этого невозможно сделать ни одним алгоритмом), но они хороши для поиска «достаточно хорошего» решения задачи «достаточно быстро». Главным же преимуществом генетических алгоритмов является то, что они могут применяться даже на сложных задачах, там, где не существует никаких специальных методов. Даже там, где хорошо работают существующие методики, можно достичь улучшения сочетанием их с генетическими алгоритмами.

Резюмируя изложенное в предыдущих разделах, можно сказать, что генетические алгоритмы лучше применять для решения следующих задач:

- **Задачи со сложным математическим представлением.** Поскольку генетическим алгоритмам нужно знать только значение функции приспособленности, их можно использовать для решения задач, в которых целевую функцию трудно или невозможно продифференцировать, задач с большим количеством параметров и задач с параметрами разных типов.
- **Задачи, не имеющие математического представления.** Генетические алгоритмы не требуют математического представления задачи. Для дальнейшего чтения 33 коль скоро можно получить значение оценки или существует метод сравнения двух решений.
- **Задачи с зашумленной окружающей средой.** Генетические алгоритмы устойчивы к зашумленным данным, например, прочитанным с датчика или основанным на оценках, сделанных человеком.
- **Задачи, в которых окружающая среда изменяется во времени.** Генетические алгоритмы могут адаптироваться к медленным

# Генетический алгоритм

изменениям окружающей среды, поскольку постоянно создают новые поколения, приспосабливающиеся к изменениям.

## Функции на языке Python для создания генетического алгоритма

```
'''
    Функция получения выжившей популяции
        Входные параметры:
        - popul - наша популяция
        - val - текущие значения
        - nsurv - количество выживших
        - reverse - указываем требуемую операцию поиска
результата: максимизация или минимизация
'''
def getSurvPopul(
        popul,
        val,
        nsurv,
        reverse
        ):
    newpopul = [] # Двумерный массив для новой популяции
    sval = sorted(val, reverse=reverse) # Сортируем
значения в val в зависимости от параметра reverse
    for i in range(nsurv): # Проходимся по циклу nsurv-раз
(в итоге в newpopul запишется nsurv-лучших показателей)
        index = val.index(sval[i]) # Получаем индекс
i-того элемента sval в исходном массиве val
        newpopul.append(popul[index]) # В новую популяцию
добавляем элемент из текущей популяции с найденным
индексом
    return newpopul, sval # Возвращаем новую популяцию (из
nsurv элементов) и сортированный список
```

# Генетический алгоритм

```
'''
    Функция получения родителей
        Входные параметры:
        - curr_popul - текущая популяция
        - nsurv - количество выживших
'''
def getParents(
        curr_popul,
        nsurv
        ):
    indexp1 = random.randint(0, nsurv - 1) # Случайный индекс первого родителя в диапазоне от 0 до nsurv - 1
    indexp2 = random.randint(0, nsurv - 1) # Случайный индекс второго родителя в диапазоне от 0 до nsurv - 1
    botp1 = curr_popul[indexp1] # Получаем первого бота-родителя по indexp1
    botp2 = curr_popul[indexp2] # Получаем второго бота-родителя по indexp2
    return botp1, botp2 # Возвращаем обоих полученных ботов



'''
    Функция смешивания (кроссинговера) двух родителей
        Входные параметры:
        - botp1 - первый бот-родитель
        - botp2 - второй бот-родитель
        - j - номер компонента бота
'''
def crossPointFrom2Parents(
        botp1,
        botp2,
        j
        ):
    pindex = random.random() # Получаем случайное число в диапазоне от 0 до 1
```

# Генетический алгоритм

```
    # Если pindex меньше 0.5, то берем значения от первого бота, иначе от второго
    if pindex < 0.5:
        x = botp1[j]
    else:
        x = botp2[j]
    return x # Возвращаем значение бота
```