КУЗНЕЦОВА О. А., ГУЩИНА О. А.

ОБЗОР ПРОГРАММНЫХ СРЕДСТВ РЕАЛИЗАЦИИ БИОНИЧЕСКИХ АЛГОРИТМОВ

Аннотация. В статье рассматриваются бионические алгоритмы и их виды. Также представлен обзор программных средств для реализации бионических алгоритмов. Продемонстрирован пример одного из бионических методов – генетический алгоритм.

Ключевые слова: бионические методы, эволюционные алгоритмы, роевой интеллект, генетический алгоритм, принципы интеллектуальных алгоритмов.

KUZNECOVA O. A., GUSHCHINA O. A.

REVIEW OF SOFTWARE FOR IMPLEMENTATION OF BIONIC ALGORITHMS

Abstract. The article discusses bionic algorithms and their types. An overview of software tools for the implementation of bionic algorithms is also presented. An example of one of the bionic methods is demonstrated - the genetic algorithm.

Keywords: bionic methods, evolutionary algorithms, swarm intelligence, genetic algorithm, principles of intelligent algorithms.

1. Суть бионических методов. На протяжении многих веков человек, взаимодействуя с окружающим его природным миром, постигал законы функционирования живых существ. Именно такие законы легли в основу создания бионических методов. Эти методы становятся мощным инструментом для различного рода задач и принятия решения. Реализация множества вычислительных алгоритмов, которые базируются на таких методах, безусловно, можно считать одним из значительных направлений развития современной науки.

На сегодняшний день учёные выделяют две группы бионических алгоритмов [1-3].

Первая группа содержит в себе алгоритмы роевого интеллекта (РИ). Это алгоритмы, которые были созданы при помощи коллективного интеллекта или методом анализа поведения группы особей (роя) сообществ животных.

Ко второй группе принадлежат эволюционные вычисления. Это алгоритмы, которые основаны на принципах «выживания сильнейших» (так называемом «естественном отборе»).

1.1. Роевой интеллект. Под термином «рой» чаще всего понимается некоторое сообщество насекомых, объёдинённых какими-либо коллективными действиями (например, рой пчёл, ос, муравьём). Каждая особь роя двигается хаотично, т.е. каждая особь обладает непредсказуемым характером поведения. Простые правила, принятые внутри роя, которые имеют значение только внутри данного сообщества и не могут использоваться единичными особями вне роя послужили базисом для создания коллективного разума. Такое явление

получило название «роевый интеллект». Такой способ мышления помогает рою максимально эффективно использовать имеющиеся ресурсы и богатства окружающей его природной среды.

Если алгоритм удовлетворяет следующим принципам, то его можно отнести к интеллектуальным алгоритмам:

– принцип разнообразия: рой хранит все свои ресурсы в нескольких рассредоточенных локациях, а не в одном месте.

– принцип адаптируемости: в случае необходимости, рой может скорректировать своё поведение;

– принцип приближения: рой может выполнять простые вычисления (пространственные и временные);

– принцип стабильности: рой после любого изменения окружающей среды никогда не отклоняться от своей линии поведения;

– принцип качества: рой всегда реагирует на различные факторы качества окружающей среды, такие как безопасность местоположения, качество пищи и тому подобные.

Главными парадигмами роевых алгоритмов являются:

– метод роя частиц;
– муравьиный алгоритм;
– алгоритм искусственного роя пчёл и другие.

1.2. Эволюционные алгоритмы. Данная группа бионических алгоритмов основывается на третьей эволюционной теории Чарльза Дарвина. Данная теория объясняет принцип естественного отбора. Её суть заключается в том, что в любом сообществе при условии ограниченности необходимых жизненно важных ресурсов неизбежно возникновение конкуренции, которая влечет за собой к преобладанию наилучших (более сильных) особей сообщества над худшими (слабыми). Результатом является то, что продолжают функционировать («выживают») наилучшие, то есть наиболее адаптированные особи. Эволюционное развитие такого сообщества методом естественного отбора позволяет поддерживать многообразие видов различных существ в живой природе и их адаптируемость к внешним условиям. Соответственно, эволюционные алгоритмы основываются на процессах и моделях естественной биологической эволюции животного и растительного миров.

Эволюционные алгоритмы распределяют свойства адаптации по средствам итерационного процесса. Этот метод является методом проб и ошибок, но для живых

организмов он максимизирует качества особей в популяции, то есть в определенном смысле составляет перспективные наборы этих особей. Перспективными не всегда являются наилучшие варианты. Часто лучшее потомство дают «обыкновенные» особи.

В генетических алгоритмах вариации решения поставленной задачи можно представить с помощью некоторой виртуальной популяции, стремящейся к наилучшему значению фитнесс-функции (целевой функцией), аналогичной выживанию в живой природе.

Процесс повторяется многократно и соответствует смене поколений, то есть популяций особей в окружающем мире. Во многих случаях происходит не только улучшение популяции, но каждой отдельной особи.

На вышеописанных механизмах строятся генетические алгоритмы. Для их реализации наиболее часто используются соответствующие генетические операторы. Основными из них являются: мутация и рекомбинация.

Эволюционные (в частности генетические) алгоритмы позволяют успешно решать большой пласт сложноразрешимых оптимизационных задач в различных проблемных областях, для решения которых плохо подходят традиционные методы. Это может быть обусловлено: неопределенностью/неточностью/многозначностью аргументов и значения фитнесс-функции, а также непрерывностью пространства поиска.

Аналогичные рассуждения можно представить и для иммунных алгоритмов.

Бионические алгоритмы редки применяются «в чистом виде». Чаще всего для повышения продуктивности решения задачи и увеличения наглядности процесса решения их совмещают с какими-либо традиционными подходами решения оптимизационных задач (например: локальный поиск).

Бионические методы чаще всего используются для гибридных задач – задач, решение которых включает совокупность нескольких разнообразных и разнотипных подзадач. Например, одни из них могут быть на нахождение локального оптимума, другие – содержат динамические ограничений так далее.

Эволюционных методы базируются на следующих основополагающих парадигмах:

- генетические алгоритмы;
- генетическое программирование;
- эволюционное программирование;
- дифференциальная эволюция.

2. Обзор программных средств реализации бионический алгоритмов. Сделаем обзор различных средств реализации бионических алгоритмов. Рассмотрим несколько самых популярных на сегодняшний день языков программирования: C++, C# и Python.

2.1. Язык C++. C++ – алгоритмический, компилируемый, статически типизированный язык программирования общего назначения. Он основывается на следующих парадигмах программирования: процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. C++ поддерживает обширную стандартную библиотеку, включающую такие возможности как: распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и многие другие особенности. C++ поддерживает свойства высокоуровневых и низкоуровневых языков программирования.

C++ широко применяется для разработки программного обеспечения. Он является самым универсальным и качественным языком программирования. Существует множество реализаций языка C++, как бесплатных, так и коммерческих. Он используется написания программного обеспечения для различных платформ (например: на платформе x86 – GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и пр.).

2.2. Язык C#. C# – объектно-ориентированный язык программирования семейства языков с C-подобным синтаксисом. Его синтаксис наиболее близок к синтаксису алгоритмических языков программирования C++ и Java. C# поддерживает статическую типизацию, поддерживает полиморфизм, перегрузку операторов и ещё большое количество полезных функции.

Переняв многое от C++, C#, опираясь на практику использования C++, исключил многие проблематичные модели при разработке программных комплексов. Например: C# в отличие от C++ не поддерживает множественное наследование классов, но допускает множественную реализацию интерфейсов.

2.3. Язык Python. Python – высокоуровневый алгоритмический язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Python является полностью объектно-ориентированным, то есть в нем всё является объектами. Синтаксис ядра языка Python минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Python является интерпретируемым языком программирования. Часто он применяется для написания скриптов.

Python является мультипарадигмальным языком программирования, поддерживающим императивное, процедурное, структурное, объектно-ориентированное программирование, метапрограммирование и функциональное программирование. Задачи обобщённого программирования часто решаются за счёт динамической типизации.

Основные архитектурные черты языка – динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений с глобальной блокировкой интерпретатора (GIL), высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Стандартная библиотека включает большой набор полезных переносимых функций, начиная от функционала для работы с текстом и заканчивая средствами для написания сетевых приложений. Дополнительные возможности могут реализовываться посредством обширного количества сторонних библиотек, а также интеграцией библиотек, написанных на Си или C++.

3. Пример реализации генетического алгоритма на языке программирования Python. Наиболее подходящим инструментом для реализации бионического алгоритма является язык программирования Python. На листинге 1 представлен пример программного кода, реализующий обобщённый генетический алгоритм.

```
from __future__ import annotations
from typing import TypeVar, Generic, List, Tuple, Callable
from enum import Enum
from random import choices, random
from heapq import nlargest
from statistics import mean
from chromosome import Chromosome

C = TypeVar('C', bound=Chromosome) #тип хромосом

class GeneticAlgorithm(Generic[C]):
    SelectionType = Enum("SelectionType", "ROULETTE TOURNAMENT")
    def __init__(self, initial_population: List[C], threshold: float, max_generations: int =
100,
                 mutation_chance: float = 0.01, crossover_chance: float = 0.7,
                 selection_type: SelectionType = SelectionType.TOURNAMENT) -> None:
        self._population: List[C] = initial_population
        self._threshold: float = threshold
        self._max_generations: int = max_generations
        self._mutation_chance: float = mutation_chance
```

```
        self._crossover_chance: float = crossover_chance
        self._selection_type: GeneticAlgorithm.SelectionType = selection_type
        self._fitness_key: Callable = type(self._population[0]).fitness
    # Используем метод рулетки с нормальным распределением,
    # чтобы выбрать двух родителей
    # Примечание: не работает при отрицательных значениях жизнеспособности
    def _pick_roulette(self, wheel: List[float]) -> Tuple[C, C]:
        return tuple(choices(self._population, weights=wheel, k=2))
    # Выбираем случайным образом num_participants и берем из них две лучших
    def _pick_tournament(self, num_participants: int) -> Tuple[C, C]:
        participants: List[C] = choices(self._population, k=num_participants)
        return tuple(nlargest(2, participants, key=self._fitness_key))
    # Замена популяции новым поколением особей
    def _reproduce_and_replace(self) -> None:
        new_population: List[C] = []
    # продолжаем, пока не заполним особями все новое поколение
        while len(new_population) < len(self._population):
        # выбор двух родителей
            if self._selection_type == GeneticAlgorithm.SelectionType.ROULETTE:
                parents: Tuple[C, C] = self._pick_roulette([x.fitness() for x in
                self._population])
            else:
                parents = self._pick_tournament(len(self._population) // 2)
        # потенциальное скрещивание двух родителей
            if random() < self._crossover_chance:
                new_population.extend(parents[0].crossover(parents[1]))
            else:
                new_population.extend(parents)
    # если число нечетное, то один лишний, поэтому удаляем его
        if len(new_population) > len(self._population):
            new_population.pop()
        self._population = new_population # заменяем ссылку
    # Каждая особь мутирует с вероятностью _titatiоп_chапce
    def _mutate(self) -> None:
```

```
            for individual in self._population:
                if random() < self._mutation_chance:
                    individual.mutate()
        # Выполнение генетического алгоритма для tax_generations итераций
        # и возвращение лучшей из найденных особей
            def run(self) -> C:
                best: C = max(self._population, key=self._fitness_key)
                for generation in range(self._max_generations):
                # ранний выход, если превышен порог
                    if best.fitness() >= self._threshold:
                        return best
                    print(f"Generation {generation} Best {best.fitness()} Avg
{mean(map(self._fitness_key, self._population))}")
                    self._reproduce_and_replace()
                    self ._mutate()
                    highest: C = max(self._population, key=self._fitness_key)
                    if highest.fitness() > best.fitness():
                        best = highest # найден новый лучший результат
        return best # лучший найденный результат из _tax_generations
```

Листинг 1. Реализация генетического алгоритма на языке Python

Представленная программа представляет собой изначальный шаблон, на основе которого можно реализовать решение любой оптимизационной задача. Для этого будет необходимо внести коррективы в нужные генетические операторы и откорректировать фитнесс-функцию. Этот шаблон прекрасно подходит для решения следующих проблем: оптимизации запросов в базах данных, задач на графах, настройки и обучения искусственной нейронной сети, задач компоновки, составления расписания, задач теории приближения, синтеза конечных автоматов и многих других задач.

Заключение. Бионические алгоритмы являются достаточно мощным инструментом современных учёных и практиков. В результате выполнения данного проекта была изучена предметная область использования алгоритмов оптимизации, основанных на принципах живой природы и процессе эволюции: рассмотрены общие принципы, лежащие в основе эволюционных вычислений и методов роевого интеллекта; выполнен обзор и проанализированы основные языки программирования для реализации бионических

алгоритмов; программно реализованы бионические алгоритмы с демонстрацией на листинге основной части генетического алгоритма на оптимальном для него языке Python.

СПИСОК ЛИТЕРАТУРЫ

1. Гладков Л. А., Курейчик В. В., Курейчик В. М., Сороколетов П. В. Биоинспирированные методы в оптимизации. – М.: ФИЗМАТЛИТ, 2009. – 384 с.
2. Семенкин Е. С., Жукова М.Н., Жуков В. Г., Панфилов И. А., Тынченко В. В. Эволюционные методы моделирования и оптимизации сложных систем. Конспект лекций. – Красноярск, 2007. – 515 с. [Электронный ресурс]. – Режим доступа: http://files.lib.sfu-kras.ru/ebibl/umkd/22/u_lectures.pdf/ (дата обращения: 16.05.2018).
3. Ashlock D. Evolutionary Computation for Modeling and Optimization. – Springer-Verlag. – Berlin. Germany, 2006. – 571 p.