Министерство науки и высшего образования Российской Федерации

Национальный исследовательский университет ИТМО

Эволюционные вычисления

Весна

2024

Лабораторная работа №6

РАСПРЕДЕЛЕННЫЕ ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ

Цель работы

Целью данной работы является освоение принципов построения распределенных и параллельных эволюционных алгоритмов для повышения их производительности и эффективности.

Краткие теоретические сведения

При возрастании сложности решаемых задач возникает необходимость в разработке более эффективных и производительных эволюционных алгоритмов, способных ускорить либо сам вычислительный процесс (производительность), либо улучшить процесс поиска оптимальных решений (эффективность). Под термином распределенные эволюционные алгоритмы стоит понимать схемы алгоритмов, которые позволяют произвести не только параллельный расчёт с целью ускорения алгоритма, но и схем, влияющих на организацию эволюции в целом. Например, распределенные эволюционные алгоритмы могут быть построены таким образом, чтобы повысить разнородность в исследовании пространства решений, увеличить статистическую устойчивость алгоритма, разбить исходную проблему на подзадачи меньшей размерности, задать особую стратегию взаимодействия решений внутри популяции.

Наиболее очевидным способом улучшения работы эволюционного алгоритма является параллельная реализация его внутренних операций. Так мы получаем **master-slave** модель эволюционного алгоритма. В основе модели лежит концепция обработки общепопуляционных операций (кроссовер, мутация, селекция) на мастер-узле, а вычисление фитнес-функции для каждого индивида осуществляется посредством передачи вычислений на ведомые узлы.

Такая модель распределенного алгоритма не влияет на логику эволюционного процесса, а только позволяет ускорить процесс вычисления. Основным узким местом такой схемы является предположение того, что вычисление фитнес-функции является наиболее вычислительно-ёмкой процедурой. Если фитнес-функция является слишком простой и быстрой операцией, то передача данных между мастером и ведомыми может занять больше времени, чем расчёт на одном узле, что может даже понизить производительность алгоритма. Однако, существуют различные решения указанного недостатка. Например, над переданным решением на ведомый узел может производится дополнительная локальная оптимизация. Еще одним недостатком схемы является синхронность. Для продолжения эволюции и перехода к следующей итерации алгоритма, мастер-узел должен получить ответ от всех ведомых узлов. При сильной вариативности времени выполнения фитнес-функции может возникнуть проблема разбалансировки вычислений. Ускорение алгоритма зависит от количества ведомых узлов и отношения затрат на передачу данных к времени выполнения фитнес функции. Схема master-slave модели представлена на рисунке 5.1.

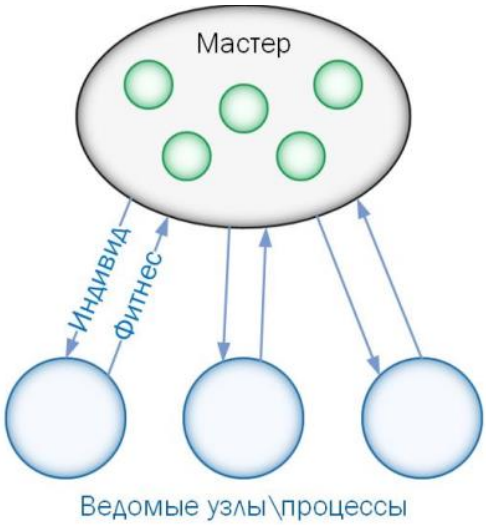

Рисунок 5.1 – Master-slave модель распределенного эволюционного алгоритма

Второй наиболее распространённой схемой распределенного алгоритма является островная **island-based** модель. Суть заключается в инициализации сразу нескольких независимых популяций (островов) с индивидами. Стратегии эволюции каждого острова могут отличаться для обеспечения разнородности поиска, а могут быть одинаковы для обеспечения лучшей устойчивости алгоритма. Предполагается, что процесс эволюции каждого острова осуществляется отдельным вычислительным узлом или процессом. Важной особенностью островной модели является дополнительная операция миграции.

Миграция подразумевает обмен индивидами между популяциями с определенным периодом времени (количеством итераций). Промежуток времени между миграциями называется эпохой. Миграция сопровождается необходимостью выбора периода эпохи и стратегии селекции мигрантов. Таким образом, островная модель может позволить как улучшить производительность за счёт параллельного вычисления нескольких популяций меньшего размера, так и изменить эффективность поиска оптимальных решений в зависимости от реализации структуры островов и выбора их параметров. Схема островной модели представлена на рисунке 5.2. В целом, островная модель меньше всего подвержена влиянию со стороны специфики решаемой задачи и довольна проста для реализации.

Клеточная **cellular** модель включает в себя одну популяцию решений, однако индивиды популяции структурированы определенным образом в виде сетки. Каждый процесс или вычислительный узел выполняет обработку определенной области сетки с индивидами, а в идеальном случае – каждый процесс обрабатывает одну клетку. Взаимодействие между индивидами (кроссовер, селекция) осуществляется только между соседними клетками на основе заданной топологии сети. В процессе селекции индивидов в рамках соседей, хорошие решения могут быть распространены в разные участки сети.

Основной идеей разработки такой схемы является реализация алгоритмов под определенные вычислительные архитектуры: суперкомпьютеры, графические процессоры, FPGA, кластеры с определенной структурой узлов. Пример клеточной модели представлен на рисунке 5.3.

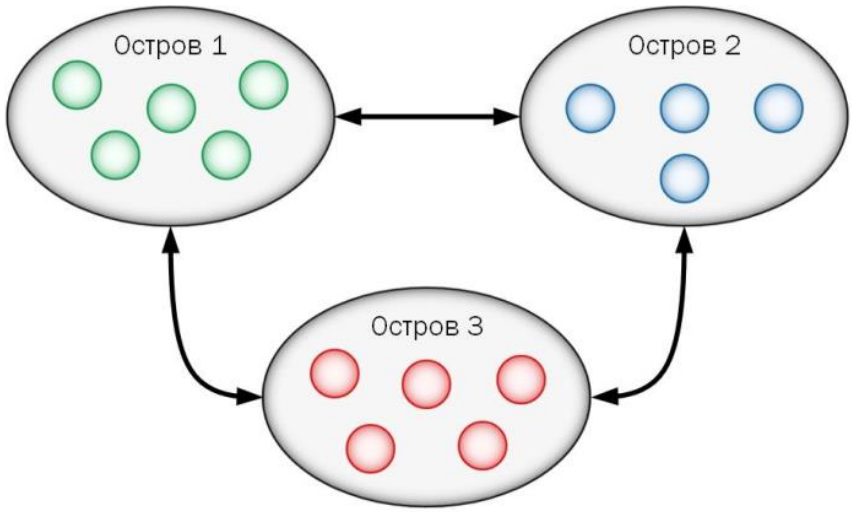

Рисунок 5.2 – Островная модель распределенного эволюционного алгоритма

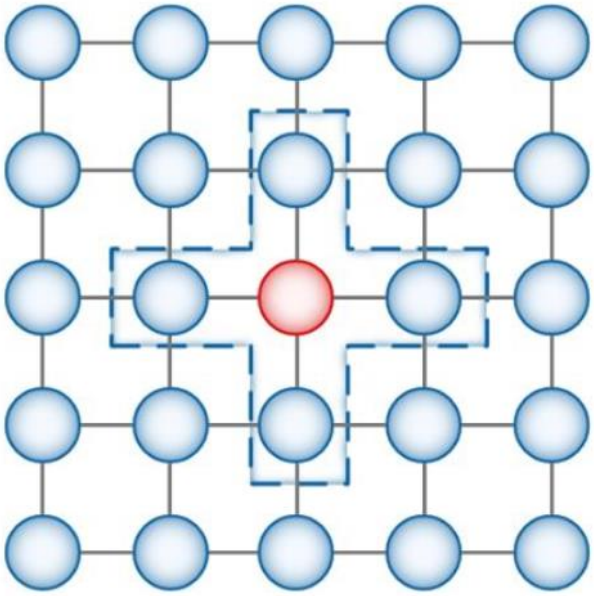

Рисунок 5.3 – Клеточная модель распределенного эволюционного алгоритма

Следующая распределенная схема направлена на уменьшение размерности исходной проблемы. Такие алгоритмы называются коэволюционными (**coevolutionary algorithms**). В основе коэволюции является разбиение решений на подзадачи меньшей размерности и их раздельная эволюция. Таким образом, коэволюция может быть рассмотрена как островная модель, где каждый остров представляет из себя эволюцию своей части от общего решения. Особенностью является этап комбинирования индивидов из разных популяций на этапе вычисления фитнес-функции для построения полноразмерных решений. Также, для реализации коэволюционной схемы необходимо выбрать способ распределения значения фитнес-функции между частями скомбинированных решений. Пример коэволюционной схемы представлен на рисунке 5.4. По большей части, коэволюция представляет из себя усложнение вычислений и тем самым зачастую приводит к уменьшению производительности алгоритма. Однако, коэволюция позволяет повысить эффективность алгоритма при высокой размерности исходной проблемы, когда поиск в пространстве решений становится неохватываемым.

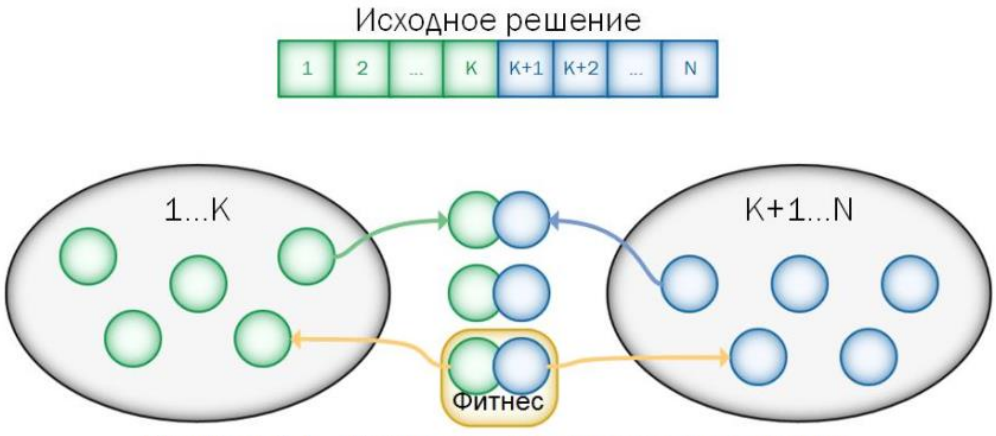

Рисунок 5.4 – Схема коэволюционного алгоритма

Все рассмотренные схемы распределенных эволюционных алгоритмов могут быть реализованы различными способами, в том числе и в виде гибридных схем (например, master-slave внутри островной модели). В зависимости от реализации и выбора параметров, целью распределенного алгоритма может являться повышение производительности или эффективности алгоритма. При разработке распределенного алгоритма необходимо учитывать специфику предметной области и вида решаемой проблемы для выбора наиболее подходящей схемы алгоритма.

Ход работы

Для выполнения данной работы могут быть использованы результаты, полученные при выполнении лабораторной работы №2. В частности, потребуется реализация функций инициализации, кроссовера и мутации для решения задачи оптимизации вещественнозначной функции. Шаблон проекта может быть загружен по ссылке:

https://gitlab.com/itmo_ec_labs/lab5

Структура проекта основана на проекте из лабораторной №2. Классы с реализацией инициализации, кроссовера и мутации остались без изменений. Класс с фитнес-функцией MultiFitnessFunction.java имеет одно отличие. Теперь фитнес-функция имеет параметр **int complexity**, который представляет собой множитель сложности вычисления фитнес-функции. Сложность заключается в повторном вычислении одной и той же функции. Таким образом, при значении параметра **complexity** = 5, время выполнения фитнес-функции приблизительно увеличится в 5 раз. В проекте присутствует два главных класса: MasterSlaveAlg.java и IslandsAlg.java. Первый является базовым алгоритмом,

https://gitlab.com/itmo_ec_labs/lab5

который по умолчанию является master-slave моделью. Второй класс соответственно представляет реализацию островной модели.

Первый этап работы. Для анализа производительности двух моделей параллельных алгоритмов необходимо получить последовательный алгоритм. Для этих целей, в классе MasterSlaveAlg.java необходимо вызвать функцию algorithm.setSingleThreaded(**true**) после создания схемы алгоритма (AbstractEvolutionEngine<**double**[]> algorithm). При передаче в функцию значения true, алгоритм будет работать в однопотоковом режиме.

Второй этап работы. Для построения островной модели необходимо создать объект IslandEvolution<**double**[]> island_model в классе IslandsAlg.java. Для этого необходимо вызвать конструктор класса с необходимым набором параметров. В качестве стратегии миграции можно выбрать реализованную в рамках фрэймворка стратегию RingMigration. Создать островную модель можно указав количество одинаковых островов либо вручную создать список необходимых объектов класса EvolutionEngine<T>. Во втором случае имеется возможность реализовать разнородные популяции с разной стратегией эволюции.

Базовый вывод информации о прогрессе эволюции уже реализован, но необходимо реализовать функционал для замера времени работы алгоритмов. Далее, необходимо провести серии экспериментов для сравнения трёх алгоритмов: однопоточный, master-slave и островной. Каждая серия экспериментов должна проводится при одинаковом размере популяции и количестве итераций для каждого алгоритма. Важно учесть, что в случае островного алгоритма размер популяции складывается по всем островам, а количество итераций равно количеству эпох (generations), умноженному на период эпохи (epochLength). Предположим, что размерность проблемы dimension = 50, а размер популяции 100.

Каждая серия экспериментов проводится для определенного значения параметра **complexity**. В каждой серии экспериментов необходимо провести минимум 10 запусков каждого алгоритма для получения усредненных значений. Для выполнения работы необходимо провести эксперименты и заполнить результаты в соответствии с таблицей 5.1 для значений **complexity** от **0** до **5**. В отчете также нужно указать все выбранные параметры алгоритмов.

Таблица 5.1 Результаты экспериментов по производительности и эффективности распределенных алгоритмов

	complexity=0		complexity=1	
	Время выполнения	Результат	Время выполнения	Результат
Single-thread				
Master-slave				
Islands				
	complexity=2		complexity=3	
…	…	…	…	…

После заполнения таблицы необходимо построить графики зависимостей времени выполнения и результата работы каждого алгоритма от значения параметра complexity.

Вопросы

1. Какая модель алгоритма лучше при каких условиях?
2. Как повлияет увеличение размерности проблемы на алгоритмы?
3. Как повлияет увеличение размера популяции?
4. Есть ли ограничение для количества островов?

Литература

1. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. – 2013.
2. Eiben A. E., Smith J. Introduction to Evolutionary Computing. – 2015.
3. Gong Y. J. et al. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art //Applied Soft Computing. – 2015. – Т. 34. – С. 286-300.