2) выявлены комбинаций генов на основе выборки реальных данных генетического обследования пациентов, полученного РНПЦ «Кардиология». Выявленные комбинации генов могут быть использованы в математической модели статистической оценки предрасположенности к АГ индивидов на основе генетических факторов и факторов риска.

Литература

1. *Pavlova O.S., Malugin V.I, Ogurtsova S.N., Novopolcev A. Yu.* Computer modeling of gene-gene and gene-environment interaction in essential hypertension // ISBRA. Minsk, 2016.
2. *Елисеева М.Р.* Молекулярно-генетические аспекты эссенциальной гипертензии. Ташкент: Шарк, 2009.
3. *Lvovs D., Фаворова О.О., Фаворов А. В.* Полигенный подход к исследованиям полигенных заболеваний // Acta Naturae. 2012. Т. 4. № 3 (14). С. 62–75.
4. *Gilks W.R., RichardsonS., Spiegelhalter D.I.* Markov Chain Monte Carlo in Practice. London: Chapman & Hall, 1996.
5. *Харин Ю.С., Малюгин В.И., Кирлица В.П., Лобач В.И., Хацкевич Г.А.* Основы имитационного и статистического моделирования. Мн.: Дизайн ПРО, 1997.
6. *Харин Ю.С., Зуев Н.М., Жук Е.Е.* Теория вероятностей, математическая и прикладная статистика. Мн.: БГУ, 2011.
7. *Favorov A. V.* A Markov Chain Monte Carlo technique for identification of combinations of allelic variants underlying complex diseases in humans / A. V. Favorov [et al.] // Genetics, 2005. Vol. 4, № 171. P. 2113–2121.

СИСТЕМА РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

М. В. Яблонская

ВВЕДЕНИЕ

На сегодняшний день существует широкий ряд задач, требующих высокой вычислительной мощности. С ростом количества сложных задач все большую популярность набирают эволюционные алгоритмы, в частности генетические алгоритмы (ГА). В задачах, требующих нахождение приближенного решения, генетические алгоритмы зачастую работают намного быстрее точных. Но, тем не менее, для нетривиальных задач выполнение ГА требует значительных вычислительных ресурсов. Одним из способов улучшения скорости вычислений ГА является использование распределенных вычислений [1].

РЕАЛИЗОВАННОЕ ПРИЛОЖЕНИЕ

В ходе работы была создана распределенная система, которая предоставляет пользователю возможность самому определить структуру осо-

бей, и производит итерационные действия алгоритма, скрывая от пользователя реализацию этих действий, но давая возможность задать некоторые параметры алгоритма (например, точность, размер популяции, способ отбора и т.д.).

Для реализации системы был выбран язык Java с его возможностями динамической загрузки классов, рефлексии и технологии сокетов. Перед началом работы с системой ее нужно сконфигурировать. Между компьютерами должна быть настроена сеть таким образом, чтобы они имели возможность взаимодействовать по протоколу TCP/IP. Для каждого экземпляра приложения прописывается информация о портах и ip-адресах остальных узлов в сети в специальный файл адресов узлов и помечается главный узел. После этого на компьютерах запускаются процессы, которые выполняют подключение к главному узлу.

Прежде чем запускать выполнение алгоритма, пользователь должен создать класс, описывающий особь. Для этого используется наследование от класса *AbstractIndivid* и определяются методы, по которым будут происходить инициализация особи, мутация, подсчет значения целевой функции, скрещивание двух особей.

На главном компьютере пользователь передает системе jar-file, содержащий класс особи и вспомогательные классы, выбирает способ распределения и задает параметры алгоритма. В системе реализованы несколько схем, по которым распределяются данные и взаимодействуют узлы системы.

Первая из схем называется “Мастер-рабочие”. Вычисление значения функции приспособленности для каждой из особей часто является самой трудоемкой операцией ГА, особенно для особей со сложными структурами. Чтобы ускорить процесс, можно использовать распределенную систему, где на одном узле (назовем его мастером) производятся операции скрещивания, мутации и отбора, а на остальных (назовем их рабочими) подсчет значений целевой функции. На каждом шаге эволюционного процесса на мастере сначала происходит деление популяции на части и каждому из рабочих посылается своя часть популяции. Деление происходит пропорционально памяти рабочих. Каждый рабочий считает значения функции приспособленности и посылает свою часть популяции обратно мастеру. Далее мастер проводит операции скрещивания, мутации и отбора.

В случае отказа какого-либо рабочего его часть популяции перераспределяется между остальными рабочими. Если выходит из строя мастер, то первый в списке адресов узлов рабочий берет его роль на себя. Остальные рабочие подключаются к нему и пересылают ему сохраненную во время последнего подсчета целевых функций часть популяции. В случае неудачи рабочие подключаются к следующему в списке узлу и т. д.

Вторая схема распределения называется “Схема островов”. В ней на узлах, соединенных в топологии "каждый с каждым", есть своя популяция небольшого размера, популяции эволюционируют независимо друг от друга. Время от времени узлы посылают друг другу лучших индивидов. При этом число итераций алгоритма уменьшается за счет расширения зоны поиска. Кроме того, снижается риск преждевременной сходимости к локальному минимуму за счет добавления новых генов в генотип популяции.

Кроме варианта, где острова развиваются по одинаковой схеме, у пользователя есть возможность выбрать схему островов с весами. В этом случае выделяется один узел, в котором не происходит мутация. На нем развивается популяция, накапливающая лучшие решения. На остальных островах расположены популяции, для которых мутация происходит с заданной пользователем частотой и количеством мутирующих за раз особей. Эти популяции нужны для исследования новых областей поиска. Однако в этом случае нужно время от времени дублировать накапливающую популяцию на остальных узлах на случай выхода из строя главного узла.

Предыдущие схемы естественным образом объединяются в “Гибридную схему”. В ней несколько популяций развиваются независимо на своей группе узлов. В каждой группе узлов выделен свой мастер и свои рабочие. В случае отказа одного из мастеров первый его рабочий становится мастером и подключается к мастерам из других групп. Остальные рабочие в данной группе подключаются к новому мастеру.

Для реализации описанных выше схем была разработана архитектура классов, представленная на рисунке 1.

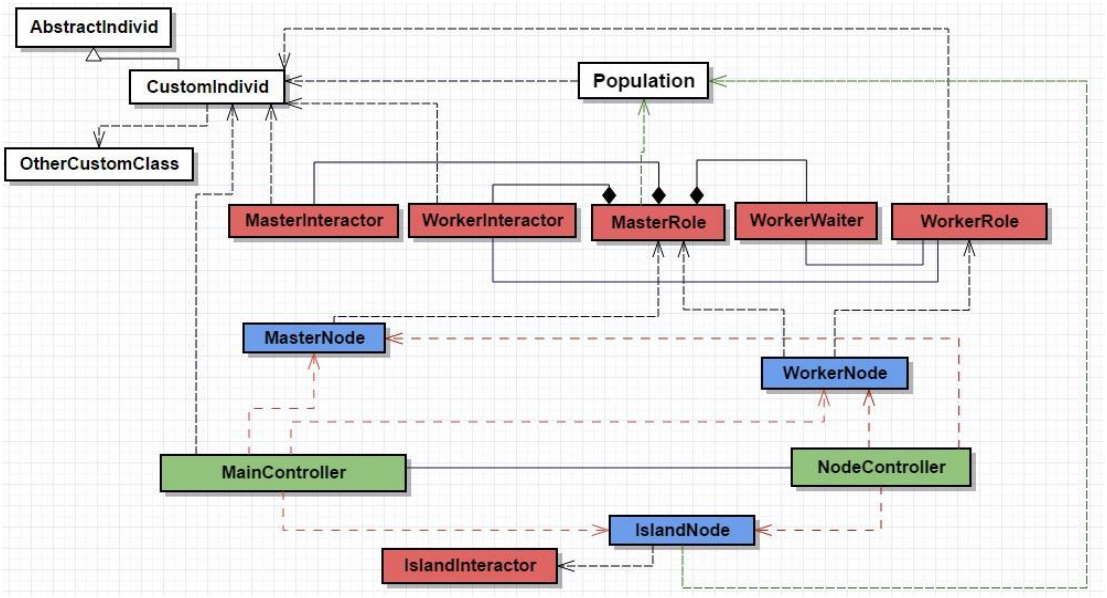

Рис.1. Архитектура классов приложения

Пользователь формирует пакет, содержащий особь и вспомогательные классы, и передает его главному модулю MainController. MainController взаимодействует с NodeController– модулем, который запускается на остальных компьютерах системы. Контроллеры сначала обмениваются данными для инициализации рабочего процесса, а затем запускают потоки MasterNode, IslandNode или WorkerNode, которые будут отвечать за организацию работы мастера, острова и рабочего соответственно в зависимости от заданных пользователем схемы и параметров алгоритма. IslandNode производит эволюционный процесс и передает свои лучшие особи другим островам, используя потоки IslandInteractor, ответственные за взаимодействие с другими островами. MasterNode запускает у себя поток MasterRole, который производит операции селекции, скрещивания и мутации. MasterRole содержит в себе поток WorkerWaiter, обрабатывающий подключения новых рабочих. Взаимодействие с каждым из рабочих осуществляется через объекты класса WorkerInteractor. Взаимодействие с другими мастерами в случае гибридной модели происходит с помощью потока MasterInteractor. WorkerNode запускает потоки WorkerRole и MasterRole. В этом случае MasterRole не производит операций ГА, а находится в процессе ожидания рабочих. WorkerRole взаимодействует с мастером, посылая данные его потокам WorkerInteractor и WorkerWaiter.

Для проверки работоспособности системы были созданы два пакета с классами, описывающими различные особи: первая – формула для аппроксимации функции по точкам, вторая – решение диофантова уравнения в виде массива значений переменных. Выбор таких особей обусловлен возможностью контролирования сложности задачи путем увеличения числа точек в функции и ее параметров в первом случае и количества неизвестных во втором случае.

В ходе тестирования системы было выявлено, что в случае отказа каких-либо узлов алгоритм продолжает работать и через некоторое время выдает корректный ответ. Для изучения скорости работы системы было запущено несколько тестов на разном количестве машин и различной точности. В данном тесте бралась особь-формула для 10 точек и двух параметров и высокая точность вычислений (разница с реальными значениями функции не более 1).

Тестирование показало, что для распределенных алгоритмов наблюдается не только улучшение скорости работы по сравнению с последовательным алгоритмом, но и уменьшение времени выполнения с увеличением числа машин. Для второй реализации особи результаты тестирования аналогичны.

ЗАКЛЮЧЕНИЕ

Таким образом, созданная система отвечает основным требованиям к распределенным системам: она реализует открытые интерфейсы, является прозрачной, т.е. представляется для пользователей как единая централизованная система, надежной и масштабируемой, т.е. производительность системы возрастает с ростом числа вычислительных узлов [2]. Полученную систему можно использовать многократно для различных задач, решаемых генетическим программированием, оптимизируя время работы.

Литература

1. *Панченко Т.В.* Генетические алгоритмы. Астрахань: Издательский дом «Астраханский университет», 2007.
2. *Косяков М.С.* Требования к распределенным системам // Введение в распределенные системы. – 3-е изд. – Санкт-Петербург: Питер, 2014. – С. 13–17