Разработка инструментальных средств поддержки эволюционных и роевых вычислений для решения задач многомерной оптимизации

Николашкин А. Г. *nagvv97@mail.ru*
Ершов Н. М. *ershovnm@gmail.com*

Применение популяционных методов

Популяционные алгоритмы предполагают одновременную обработку нескольких вариантов решения задачи оптимизации, т.е. оперируют *популяцией* решений. К ним относятся *эволюционные алгоритмы* и *алгоритмы роевой оптимизации*.

Такие алгоритмы эффективнее в решении мультимодальных и многомерных задач оптимизации, чем классические “траекторные” методы оптимизации.

Примеры популяционных алгоритмов

Эволюционные алгоритмы:

- Генетические алгоритмы
- Метод дифференциальной эволюции
- Алгоритм эволюционной стратегии
- и т. п.

Алгоритмы роевой оптимизации

- Метод роя частиц
- Алгоритм поиска алгоритм
- Пчелиный алгоритм
- Алгоритм гравитационного поиска
- и т. п.

Существующие решения

Название	ЯП	Параллелизация	Инструменты
A parallel global multiobjective framework for optimization: pagmo2 / pygmo2	C++ / Python	evaluate, островная модель	TBB, custom / multiprocessing, ipyparallel, custom
Distributed Evolutionary Algorithms in Python (DEAP)	Python	evaluate, островная модель	multiprocessing, SCOOP, custom
jMetal: a framework for multi-objective optimization with metaheuristics	Java	evaluate	threads, Apache Spark, custom
pymoo: Multi-objective Optimization in Python	Python	evaluate	multiprocessing, dask, custom
Paradiseo: a Heuristic Optimization Framework	C++	evaluate, островная модель, операторы	OpenMP, std::thread, MPI

Пример: pagmo2

Написан на языке программирования *C++*. Использует Type Erasure.

Алгоритмов из коробки:

- глобальной оптимизации: Ant Colony Optimization, Differential Evolution, Particle Swarm Optimization, Simple Genetic Algorithm, Artificial Bee Colony, ...
- локальной оптимизации из NLopt, Ipopt и т.п
- мета-алгоритмы: Monotonic Basin Hopping, Cstrs Self-Adaptive, Augmented Lagrangian method

Имеет широкий набор готовых тестовых задач.

Механизмы параллелизации:

- Островная модель - мультипроцессинг, потоки, ipyparallel(pygmo)
- Параллелизация вычисления целевой функции - потоки, ipyparallel(pygmo), реализованная задачей

Пример: jmetal

Написан на языке программирования *Java*.

Больше нацелен на многоцелевую оптимизацию.

Алгоритмов из коробки:

- Genetic Algorithm, Evolution Strategy, Differential Evolution, CMA-ES, Particle Swarm Optimization, Coral Reef Optimization
- NSGA-II, SPEA2, PAES, PESA-II, OMOPSO, MOCell, AbYSS, MOEA/D, …

Имеет набор готовый тестовых задач.

Механизм параллелизации:

- Интерфейс *SolutionListEvaluator* для распараллеливания вычисления целевой функции

Реализации: *Sequential*, *Multithreaded*, *Spark*

Небольшое резюме по существующим решениям

- Не все решения предлагают возможность разработки собственных алгоритмов
- Не все решения предлагают из коробки готовый набор алгоритмов
- Ограничиваются распараллеливанием вычисления целевой функции
- Предлагают ограниченный набор инструментов распределения вычисления

Библиотека Insectae

Язык программирования - Python

Цели:

- Возможность структурной и параметрической настройки существующих алгоритмов
- Стандартизированные средства для разработки новых алгоритмов
- Эффективная параллелизация вычислений
- Тестирующая подсистема для выбора наиболее эффективного алгоритма

Модель организации алгоритма

Общие свойства популяционных алгоритмов:

- Постановка задачи
  - Задана некоторое пространство решений
  - Определена целевая функция
  - Необходимо найти минимум\максимум
- Схожая структура
  - Имеется популяция пробных решений
  - Итерационное развитие популяции
  - Некоторое условие останова
- Схожие паттерны взаимодействия между особями

Модель организации алгоритма - данные

Типы данных:

- Локальные данные - принадлежат конкретной особи
  - Положение, скорость, значение целевой функции и т.п.
- Глобальные данные - принадлежат общей среде
  - Целевая функция, направление оптимизации, параметры и т.п.

Доступ к данным у особей:

- К данным среды - прямой
- К своим данным - прямой
- К данным других особей - через специальные процедуры

Модель организации алгоритма - выполнение

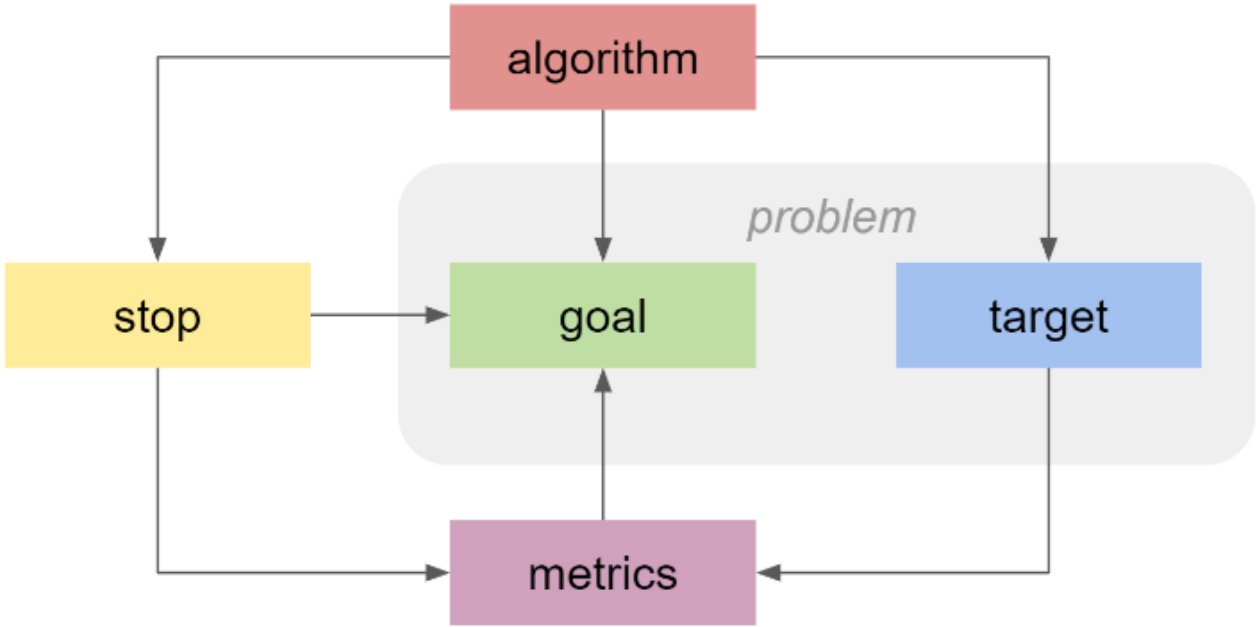
Выполнение популяционного алгоритма происходит в три этапа:

1) *Инициализация*: выполняется создание популяции, настройка параметров особей и инициализация параметров среды.
2) *Основной цикл*: на каждой итерации применяется определенная алгоритмом последовательность операторов и проверяется условие останова.
3) *Завершение:* производится финальная обработка полученных данных.

Модель организации алгоритма - взаимодействие

- *evaluate* - вычисление целевой функции для каждой особи
- *foreach* - выполнение некой функции над каждой особью
- *neighbors* - выполнение некой функции над парами смежных по перестановке особей
- *pairs* - выполнение некой функции над соответствующими парами из двух заданных популяций
- *pop2ind* - выполнение некой функции над каждой особью в паре с заданной популяцией
- *reduce* - выполнение редукции над некоторыми данными из особей
- *signals* - вычисление “сигналов” между особями

Ядро библиотеки



Представление задач

Problem:

- *Goal* - представляет направление оптимизации и определяет метод сравнения решений
- *Target* - представляет целевую функцию, определяет типа задачи и базовую начальную инициализацию решений

Stop - определяет условие останова

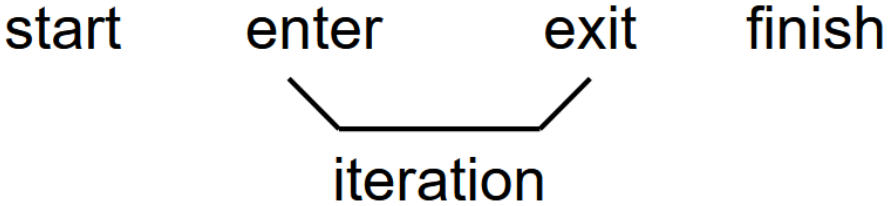
Metrics - реализует сбор статистики

Представление алгоритма

Базовый класс *Algorithm*:

- Хранит популяцию решений и данные среды.
- Отвечает за выполнение алгоритма.
- Предлагает интерфейс к паттернам взаимодействия.
- Позволяет выполнить структурную настройку алгоритма.

Точки выполнения пользовательских процедур:



Способы параллелизации

- Параллелизация работы алгоритма над одной популяцией
  - как правило вычисление целевой функции
- Островная модель
- Параллелизация независимых выполнений алгоритмов
  - в рамках тестовой системы
  - в рамках оптимизации метапараметров

Параллельная обработка одной популяции

Вычисление целевой функции - $fn(x\_i) \rightarrow f\_i$; паттерн *evaluate*

Интерфейс *map/starmap*: fn, [x0, x1,...] → futures[f0, f1, …]

Доступна в том или ином виде в библиотеках: multiprocessing, mpi4py, dask, scoop, apache spark, ray, и т.п.

Специальный класс-обертка *Executor* позволяет:

- Нивелировать различия в интерфейсах
- Выставлять параметры (например *chunksize*)
- Использовать специфичную логику

Итого

- Разработан прототип библиотеки поддержки эволюционных и роевых вычислений на Python3
- Добавлена поддержка параллелизации вычисления целевой функции

Планы:

- Расширение возможностей параллелизации
- Создание тестовой системы
- …

Спасибо за внимание