

Описание Математического Бота

Латыпов Владимир Витальевич

28 июля 2021 г.

Содержание

1	Формулировка задачи	3
2	Обзор	3
3	Технические подробности	4
4	Описание алгоритма в общих чертах	4
4.1	Разбор выражений	4
4.2	Дерево алгоритмов оптимизации	4
4.2.1	Идея	4
4.2.2	Распределение вычислительного ресурса	4
5	Модификации в ГА	5
6	Дальнейшее развитие	6

1. Формулировка задачи

Математический бот — это набор инструментов для работы с математическими сущностями через интерфейс привычных нам социальных сетей. В отличие от WolframAlpha, бот специализируется на оптимизации многомерных функций и численном решении уравнений с большим количеством переменных (проверено, что с этим бот справляется лучше).

Технически бот состоит из двух частей:

- Условный «front-end» — программа (написана на python), которая через vk_api выполняет коммуникативную функцию, то есть общается с пользователем и, когда требуется, вызывает основной блок.
- А именно — «back-end» интерфейс, который написан на C++ и компилируется в полноценное консольное приложение. Он и занимается оптимизацией функций, решениями уравнений и т.д.

Поэтому больший интерес, конечно, представляет back-end часть.

2. Обзор

Бот умеет:

- Оптимизировать функции
- Решать уравнения
- Строить графики

Третий режим добавлен для удобства — чтобы рассматривать функции (правда, пока только 2-х мерные), не выходя из диалога с ботом.

Во всех случаях вводится математическое выражение и, возможно, дополнительные параметры. Предусмотренно два варианта введения запросов боту:

- Через коммуникацию: от пользователя требуется читать сообщения бота, предложенные ответы, знакомясь с интерфейсом бота, и, отвечать на них (зачастую достаточно выбрать вариант из предложенных). Этот вариант подходит для людей, которые только знакомятся с возможностями бота.

- Quick Input Mode (QIM). Если пользователь точно знает, что ему нужно и что значат аргументы, для него целесообразно использовать эту возможность. Вся информация для запроса передаётся одним сообщением, содержащим корректную команду QIM.

Типичный запрос выглядит так (переносы строки не важны):

```
optimize x + y^2 - 1000.5
for x in [-10; 100],
y in (10.3e-100, 123)
| minorant 10
```

Подробнее об интерфейсе можно прочитать в Инструкции (ВСТАВИТЬ ССЫЛКУ!).

В отличие от других подобных инструментов, бот предоставляет дружелюбный интерфейс: если пользователь что-то некорректно указал, бот известит его об этом. Однако иногда бот проверяет характер: например, ему не очень нравится, когда много раз неправильно указывают аргумент или когда пишут с маленькой буквы.

3. Технические подробности

4. Описание алгоритма в общих чертах

4.1. Разбор выражений

Оптимизация: если какая-то ветка полностью вычислима заранее (без знания переменных), она считается сразу. Если $+0$ или $*1$, это тоже убирается.

4.2. Дерево алгоритмов оптимизации

4.2.1. Идея

4.2.2. Распределение вычислительного ресурса

В случае МатБота результаты профайлинга подтверждают предположение о том, что бо́льшая часть вычислительного времени ($\gg 95\%$) используется именно для подсчёта функции ошибки, а не для операций с геномами.

В условиях, когда выделенное количество вычислительных ресурсов может быть разным для разных людей и запросов, особенно остро встаёт вопрос о распределении этих вычислительных ресурсов.

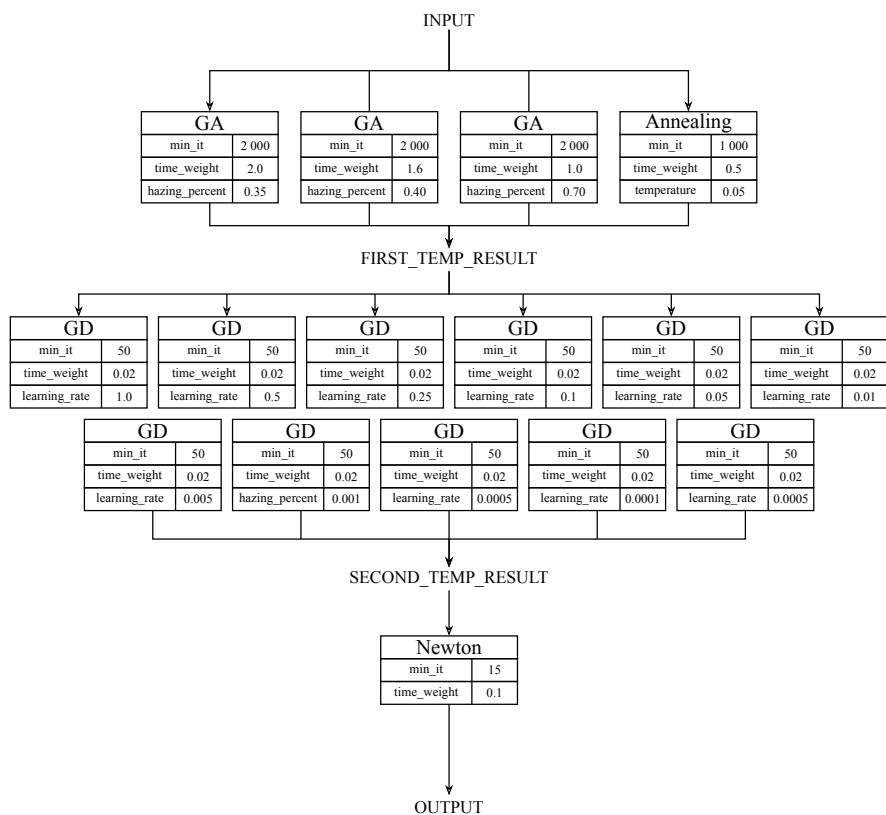


Рис. 1: Схема дерева оптимизации

5. Модификации в ГА

Подробное описание ГА (в том числе — моей версии) можно найти в описании робота-художника в соответствующей секции. Здесь сосредоточусь на том, что относится конкретно к боту.

В секции 4.2.2 описано, как вычислительный ресурс распределяется между разными алгоритмами. Однако на том моменте, когда стало

известно количество раз, сколько можно вычислить функцию ошибки в течение времени выполнения ГА, вопрос ещё не закрыт.

Как известно, генетический алгоритм в каждую из E «эпох» вычисляет функцию ошибки для $population_size$ геномов. То есть суммарное количество вычислений: $computational_resource = E \times population_size$.

Встаёт вопрос, какое соотношение значений E и $population_size$ обеспечивает наилучшую работу алгоритма.

Я решил проверить это экспериментально. Изначальное предположение состоит в том, что разумно распределять ресурс так:

$$\begin{cases} E = computational_resource^{epoch_pow} \\ population_size = \frac{computational_resource}{E} = computational_resource^{1-epoch_pow} \end{cases} \quad (1)$$

Вопрос был в том, какое $epoch_pow$ выбрать. Для решения вопроса я выбрал функцию с более или менее сложным рельефом и для разного количества итераций построил графики зависимости достигаемого $fitness$ (значения целевой $fitness$ -функции) от $epoch_pow$.

Конечно, в целях помехоустойчивости для каждого $epoch_pow$ делалось далеко не одно измерение. Важно оценивать не только минимальное, или только максимальное, или только некое среднее (иначе можно упустить важную информацию о недостатках или достоинствах того или иного $epoch_pow$).

Поэтому для комплексной оценки я посчитал «нечёткие» верхнюю и нижнюю границы множества значений $fitness$ -а для нескольких попыток запуска. Под нечёткой нижней границей понимается такое число, которое бы оценило «почти худший случай работы алгоритма». С верхней границей — аналогично (почти лучший).

Для их оценки я решил использовать вариацию среднего Колмогорова:

$$M(x_1, \dots, x_n) = \varphi^{-1} \left(\frac{1}{n} \sum_{k=1}^n \varphi(x_k) \right) \quad (2)$$

, где (в моём случае):

для нижней границы $\varphi(x) = x^{-1.5}$ (это повышает чувствительность к малым значениям), для верхней границы $\varphi(x) = x^{2.5}$ (это повышает чувствительность к большим значениям).

Результаты таковы:

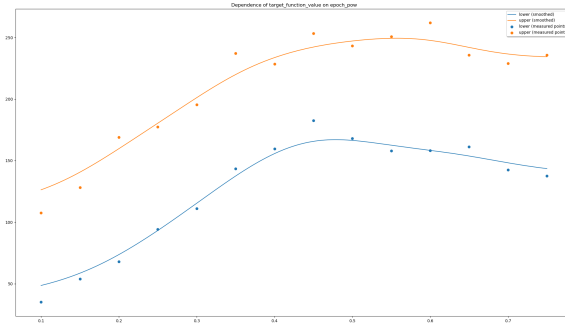


Рис. 2: График для 1'000 вычислений

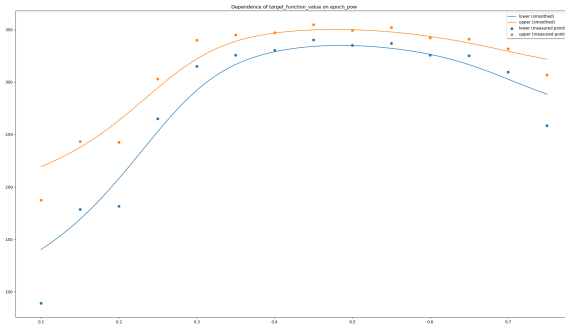


Рис. 3: График для 5'000 вычислений

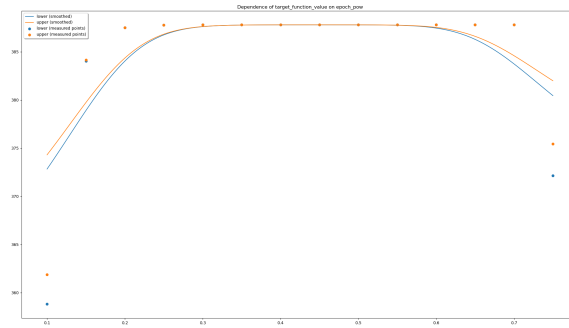


Рис. 4: График для 100'000 итераций

6. Дальнейшее развитие

— Добавить ещё алгоритмы. — Использовать аналитические методы, где возможно (например, для решения уравнений)