Генерация тестов для задачи поиска максимального потока с использованием эволюционных алгоритмов и матричного представления графа

Миронович В. А., магистрант кафедры компьютерных технологий Университета ИТМО, mironovichvladimir@gmail.com

Буздалов М. В., к.т.н., доцент кафедры компьютерных технологий Университета ИТМО, mbuzdalov@gmail.com

Аннотация

Результаты теоретических исследований в области анализа вычислительной сложности задач начинают использоваться для разработки новых эволюционных алгоритмов. В то же время сложно понять, какие из теоретических идей имеет смысл применять при работе с практическими задачами.

Ранее мы использовали модифицированный $(1 + (\lambda, \lambda))$ алгоритм, основанный на теоретических результатах анализа вычислительной сложности задач, для генерации сложных тестов для задачи поиска максимального потока. Результаты полученные с помощью данного алгоритма были хуже, чем полученные (1+1) эволюционным алгоритмом и одним из вариантов генетического алгоритма. Причиной низкой эффективности алгоритма $(1+(\lambda, \lambda))$ могло послужить не типичное для него представление графа в виде списка ребер.

В данной работе мы исследовали представление графа в виде матрицы смежности, которое легко преобразуется в битовую строку, если пропускные способности имеют фиксированный размер в битах. Несмотря на то, что данное представление лучше подходит для алгоритма $(1+(\lambda, \lambda))$, результаты экспериментов показали, что данный алгоритм работает хуже всех остальных алгоритмов.

Основываясь на другой идее из теории вычислительной сложности, мы разработали алгоритм XOR3-DE. За основу алгоритма взята дифференциальная эволюция, но для рекомбинации особей используется тернарный оператор XOR. Разработанный алгоритм показывает хорошие результаты, немного проигрывающие лишь лучшим генетическим алгоритмам.

Введение

Большинство эволюционных алгоритмов практически не используют информацию, полученную при рассмотрении “слабых” решений. Наиболее простой (1+1) эволюционный алгоритм полностью игнорирует решения, значение функции приспособленности которых меньше, чем у родителя. Чем ближе алгоритм подходит к оптимуму, тем больше информации он теряет.

Недавно предложенный алгоритм $(1 + (\lambda, \lambda))$ был разработан для того, чтобы побороть эту проблему [1]. За счет увеличения вероятности мутации, алгоритм быстрее исследует пространство поиска, но чаще получает особи со значением функции приспособленности меньше, чем у родителя. Независимо от значения функции приспособленности, полученные в результате мутации особи используются с оператором скрещивания. Таким образом, алгоритм в среднем использует больше информации, полученной при вычислении функции приспособленности. Как следствие, алгоритм $(1+(\lambda, \lambda))$ работает быстрее, чем алгоритмы $(\mu + \lambda)$ и (μ, λ) на задаче OneMax.

Авторы алгоритма $(1 + (\lambda, \lambda))$ предложили проверить его работоспособность на практических задачах. В предыдущей работе [2] мы применили модифицированный алгоритм $(1+(\lambda, \lambda))$ для генерации тестов к задаче поиска максимального потока. На данной задаче алгоритм уступает генетическому алгоритму, и алгоритму (1+1), показавшими хорошие результаты [3]. Одной из возможных причин является тот факт, что, в отличие от задачи OneMax, в которой особи представляются битовыми строками, особи для генерации тестов для задачи о максимальном потоке были представлены в виде списка ребер. Данное представление не типично для исходного алгоритма $(1+(\lambda, \lambda))$, что могло послужить причиной его малой эффективности.

В данной работе исползуется другое представление графа, матрицы смежности. Данное представление позволяет с помощью небольшой модификации представить граф в виде битовой строки, что лучше подходит для использования в алгоритме $(1 + (\lambda, \lambda))$.

Эксперименты показали, что с использованием данного представления алгоритм $(1 + (\lambda, \lambda))$ показывает очень плохие результаты, проигрывая не только вариантам генетических алгоритмов и алгоритму (1+1), но и другим вариациям самого себя, не использующим битовое представление графа. В то же время, мы разработали новый алгоритм, основанный на идеях из теоретического анализа вычислительной сложности задач. Данный алгоритм использует тернарный оператор XOR при скрещивании особей и по структуре схож с алгоритмами дифференциальной эволюции. Он показал хорошие результаты при генерации тестов для задачи о максимальном потоке, немного проигрывающие лишь лучшим генетическим алгоритмам.

Задача о максимальном потоке

Задача о максимальном потоке [4] формулируется следующим образом: для заданных графа $G = (V, E)$ и пропускных способностей c_i для каждого ребра, и двух вершин, обозначенных s – исток и t – сток, необходимо найти *максимальный поток*.

Максимальный поток это такой набор чисел f_i, что:

- для всех ребер, f_i это неотрицательное число не превышающее значение пропускной способности c_i для данного ребра,
- для всех вершин, кроме s и t, сумма f_i по исходящим ребрам равна сумме по входящим,
- для вершины s разность сумм f_i по исходящим и входящим ребрам максимальна.

Существует множество решений для задачи поиска максимального потока [5, 6, 7, 8, 9]. Большинство алгоритмов способны решать случайно сгенерированные экземпляры задач достаточно быстро, однако асимптотические оценки максимального времени работы данных алгоритмов предполагают существование сложных экземпляров задач.

Из всех алгоритмов решения задачи о максимальном потоке в данной работе мы рассматриваем два: алгоритм Диница [7] и улучшенный алгоритм поиска кратчайших путей (ISP) [8]. Эти алгоритмы использовались ранее [3, 2], так как они показывают высокую эффективность и разное поведение на похожих экземплярах задач.

Алгоритмы

В данной работе в качестве задачи оптимизации рассматривается генерация сложных экземпляров задачи о максимальном потоке (сложных тестов). Цель генерации тестов – максимизация времени работы алгоритма на полученном тесте.

Экземпляр задачи, а также особь эволюционного алгоритма, это граф, представленный матрицей смежности M. M это матрица размера $|V| \times |V|$, в которой каждый элемент $M_{i,j}$ представляет пропускную способность ребра, соединяющего вершину i и j. Ввиду отсутствия ребер для которых $i < j$, как и в [3], M это верхнетреугольная матрица.

В данной работе мы рассматриваем следующие модификации известных алгоритмов оптимизации:

- Стандартный (1+1) эволюционный алгоритм.
- Две версии $(1+(\lambda,\lambda))$ эволюционного алгоритма. Одна из них является модификацией алгоритма из [2] для работы с матричным представлением графа, другая является реализацией алгоритма из [1] работающая с графом как с битовой строкой.
- Стандартный (1+λ) эволюционный алгоритм.
- Две версии генетического алгоритма, различающиеся оператором скрещивания. В одной используется одноточечный оператор скрещивания, в другой – однородный.

Листинг 1 Алгоритм XOR3-DE

```
1:  Population P ← N randomly generated individuals
2:  while stopping criterion is not reached do
3:      while |P'| < N do
4:          Randomly select individuals x_1, x_2 and x_3 from P
5:          Generate individual x' such that M'^{x'}_{i,j} ← M^{x_1}_{i,j} ⊕ M^{x_2}_{i,j} ⊕ M^{x_3}_{i,j}
6:          P' ← MUTATE x' (flip bits with probability 1/(|E| · k))
7:      end while
8:      for all i ∈ [1, N] do
9:          x_i ← P, x'_i ← P'
10:         if f(x_i) ≤ f(x'_i) then
11:             P'' ← x'_i
12:         else
13:             P'' ← x_i
14:         end if
15:     end for
16:     P ← P''
17: end while
```

Также предложен новый алгоритм XOR3-DE, основанный на принципах дифференциальной эволюции [10]. Псевдокод алгоритма XOR3-DE представлен на листинге 1. В данном алгоритме в качестве оператора скрещивания используется тернарный оператор XOR, работающий на битовом представлении графа. После применения оператора XOR алгоритм использует

оператор мутации на полученных особях и производит селекцию схожую с используемой в классической дифференциальной эволюции.

Исходное поколение для каждого из алгоритмов составляется из случайно сгенерированных особей. Верхняя граница на пропускную способность ребра C выбирается равной 2^m, для некоторой целой константы m. В противном случае потребовалась бы дополнительная фаза для ограничения пропускных способностей полученных при применении битовых операций.

Функция приспособленности – число ребер посещенных алгоритмом в процессе поиска максимального потока. Эта функция пропорциональна времени работы алгоритма, и наиболее эффективна при использовании в оптимизации с ограниченным бюджетом [3].

Результаты

Таблица 1: Результаты запусков с использованием алгоритма Диница

	1+1	1+(25,25)	1+(2×8)	BS 1+(8,8)	GA+SP	GA+UF	XOR3-DE
MIN	126989	79574	103812	42019	289514	183101	248642
MEDIAN	264723	278065	298243	131284	411630	464155	405519
AVG	301515	310605	289123	125979	445204	453570	413826
MAX	644428	627050	524176	193368	749243	593996	678650

Таблица 2: Результаты запусков с использованием алгоритма ISP

	1+1	1+(25,25)	1+2×16	BS 1+(8,8)	GA+SP	GA+UF	XOR3-DE
MIN	78483	353076	194868	32519	623280	542586	474092
MEDIAN	629558	668441	619181	321418	734376	739333	654022
AVG	614957	667229	604203	245837	739030	736445	682065
MAX	970195	939304	884638	381000	882934	848783	952571

Как и в предыдущих работах [3, 2] для проведения экспериментов максимальное число вершин $V = 100$, число ребер $E = 5000$. Максимальная пропускная способность C равна $2^{13} = 8192$. Для каждого алгоритма решения задачи о максимальном потоке и оптимизационного алгоритма было выполнено 30 запусков с бюджетом вычисления функции приспособленности 500000.

Полученные максимальные, минимальные, средние и медианные значения функции приспособленности для алгоритмов представлены на таблицах 1 и 2. Для алгоритмов $(1 + (\lambda, \lambda))$ и (1+λ), где использовались различные

значения параметра λ, представлены данные для запусков с наилучшими результатами.

В обоих случаях алгоритм $(1 + (\lambda, \lambda))$, работающий с графом как с битовой строкой, оказывается неэффективным. Обычная версия алгоритма $(1 + (\lambda, \lambda))$ показывает результаты близкие к результатам алгоритмов (1+1) и (1+λ). Предложенный алгоритм XOR3-DE показывает хорошие результаты, сравнимые с результатами генетических алгоритмов.

Заключение

Мы провели эксперименты по изучению поведения эволюционных алгоритмов при генерации тестов для задачи о максимальном потоке с использованием представления графа в виде матрицы смежности. Результаты показали, что алгоритм $(1 + (\lambda, \lambda))$ имеет низкую эффективность, однако предложенный нами алгоритм XOR3-DE показал хорошие результаты сравнимые с результатами лучших алгоритмов – генетических.

Литература

[1] B. Doerr, C. Doerr, and F. Ebel. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567:87–104, 2015.

[2] V. Mironovich and M. Buzdalov. Hard test generation for maximum flow algorithms with the fast crossover-based evolutionary algorithm. In *Proceedings of Genetic and Evolutionary Computation Conference Companion*, pages 1229–1232, 2015.

[3] M. Buzdalov and A. Shalyto. Hard test generation for augmenting path maximum flow algorithms using genetic algorithms: Revisited. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 2121–2128, 2015.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd Ed.* MIT Press, Cambridge, Massachusetts, 2001.

[5] L. R. Ford Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[6] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–262, 1972.

[7] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.*, 11(5):1277–1280, 1970.

[8] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[9] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 136–146, New York, NY, USA, 1986. ACM.

[10] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.