Университет ИТМО

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Кевер Михаил Еневич

Теоретическое исследование вычислительной сложности задачи Jump-K

Научный руководитель: декан ФИТиП, д. т. н., проф. В. Г. Парфенов

Санкт-Петербург
2014

Оглавление

Список алгоритмов

Введение

Теория сложности ставит своей целью определение сложности вычислительных задач. В классической теоретической информатике плодотворное взаимодействие теории сложности, изучающей, какие минимальные вычислительные затраты необходимы для решения задачи, и теории алгоритмов, дающей решение и тем самым показывающей, каких вычислений достаточно, является одной из ключевых сил, развивающих область.

В случае эволюционных алгоритмов и других методов вероятностного поиска уже существует немало результатов со стороны теории алгоритмов. В таких книгах, как [Jan13, NW10, AAD11], можно найти множество примеров того, как анализуется время работы для различных задач и алгоритмов для их решения.

Со стороны теории сложности же в то же время существует некоторое отставание, и классические спосообы оценки сложности плохо подходят для вероятностных эвристик. Отличие изучаемого нами случая в том, что алгоритм не имеет полных знаний о решаемой задаче либо потому, что их просто нет, либо же они есть, но их неудобно использовать, например, из-за их сложности. Вместо этого, алгоритм получает информацию о задаче с помощью вычисления оценочной функции (функции приспособленности) в неких точках.

Black-box сложность — это попытка создать теорию сложности для методов вероятностного поиска. Впервые она была введена в 2006 году в работе [DJW]. Модель алгоритмов поиска, определенную в этой работе, также называют неограниченной моделью, поскольку она очень слабо ограничивает то, какую информацию получает алгоритм, и как он может ее использовать. Как следствие, эта модель допускает очень специализированные алгоритмы, и оценки сложности задач получаются ниже, чем дают стандартные методы

поиска.

Это расхождение, как одна из причин, мотивировало создание альтернативной модели, описанной в [LW10]. В этой модели, называемой непредвзятой, алгоритм может получать точки только используя так называемые непредвзятые операторы. В оригинальной работе рассматривается ряд задач для случая унарных операторов, покрывая тем самым алгоритмы, получающие новые запросы только путем изменения одного из предыдущих запросов.

В работе [DJK+11] исследуется сложность оптимизации хорошо изученных функций OneMax и LeadingOnes в случае непредвзятых операторов более высокого порядка. Полученные оценки оказываются ниже, чем для унарных операторов, что означает, что такие алгоритмы более сильны, и, например, дает право говорить о преимуществе эволюционных алгоритмов, использующих операции скрещивания, поскольку они могут давать более низкие оценки, чем теоретически возможно без их использования.

В данной работе мы рассмотрим класс функций, известный как Jump, с точки зрения обеих моделей, и получим как верхние, так и нижние оценки на сложность. В случае неограниченной модели полученные нижняя и верхняя оценка позволят полностью закрыть вопрос об асимптотической сложности, а в ряде случаев совпадать будет не просто асимптотика, но и явные значения сложности с точностью до членов меньшего порядка. Для непредвзятой же модели будет получена достаточно точная верхняя оценка в случае операторов размерности не меньше трех и любых параметров класса Jump.

Алгоритмы и доказательства оценок в этой работе будут основаны как на уже известных решениях для других задач, так и на новых идеях, которые, как можно надеяться, смогут помочь в получении оценок и для других классов функций.

Глава 1. Основные определения

1.1. Black-box сложность

В этом подразделе мы формально определим две black-box модели, введенные в [DJW] и в [LW10]. Первую модель мы назовем *неограниченной* моделью, а вторую — *непредвзятой* моделью. Каждая модель определяет класс алгоритмов. Сложность класса функций, таким образом, определяется относительно алгоритмов в соответствующей модели.

В обеих моделях мы имеем дело с классом функций $\mathcal{F}$, известным алгоритму. Фиксируется некая функция f, принадлежащая классу, и скрывается от алгоритма. Алгоритм может получать информацию об f путем запросов к оракулу о значениях функции в точках. Цель алгоритма состоит в том, чтобы найти некое глобальное оптимальное значение функции. Без потери общности, можно рассматривать максимизацию в качестве цели оптимизации. Две рассматриваемые модели различаются в информации, доступной алгоритму, и точках, которые можно запрашивать.

В этой работе мы будем рассматривать задачу максимизации функций $f : \{0,1\}^n \to \mathbb{R}$. *Неограниченная* модель рассматривает все алгоритмы, общий вид которых описан как алгоритм 1. Особенность этой модели в том, что она не накладывает никаких ограничений на допустимые запросы, поэтому в ней содержится наиболее широкий класс алгоритмов.

Чтобы исключить алгоритмы, поведение которых сильно отличается от естественных эвристик, используемых в оптимизации, можно ввести дополнительные ограничения. *Непредвзятая* модель, введенная в [LW10], вводит два существенных ограничения. Во-первых, решения, принимаемые алгоритмом, могут зависеть только от наблюдаемых значений функции, но не от самих точек. Во-вторых, алгоритм может делать только запросы, полученные при помощи непредвзятых операторов, определенных ниже.

Алгоритм 1: Black-box алгоритм в неограниченной модели

Выбрать вероятностное распределение p^0, заданное на $\{0,1\}^n$;
Выбрать x^0 из p^0 и запросить $f\left(x^0\right)$;
for $t = 1, 2, 3, \ldots$ *до окончания работы* **do**
  В зависимости от $\left(x^0, f\left(x^0\right)\right), \ldots, \left(x^{t-1}, f\left(x^{t-1}\right)\right)$ выбрать вероятностное распределение p^t, заданное на $\{0,1\}^n$;
  Выбрать x^t из p^t и запросить $f\left(x^t\right)$;
end

Определение 1 (Непредвзятый оператор порядка k [LW10])**.** *Зафиксируем* $k \in \mathbb{N}$. *Непредвзятым распределением порядка* k $D\left(\cdot \mid x^1, \ldots, x^k\right)$ *называется такое вероятностное распределение над* $\{0,1\}^n$*, что для любых* $y, z \in \{0,1\}^n$ *и перестановки* σ *размера* n *выполняются следующие два условия:*

1. $D\left(y \mid x^1, \ldots, x^k\right) = D\left(y \oplus z \mid x^1 \oplus z, \ldots, x^k \oplus z\right)$;
2. $D\left(y \mid x^1, \ldots, x^k\right) = D\left(\sigma\left(y\right) \mid \sigma\left(x^1\right), \ldots, \sigma\left(x^k\right)\right)$.

Первое свойство в определении называется инвариантностью относительно побитового исключающего или, второе — инвариантностью относительно перестановки. Непредвзятая модель порядка k содержит все алгоритмы, имеющие вид алгоритма 2.

Алгоритм 2: Black-box алгоритм в непредвзятой модели

Выбрать x^0 равномерно случайно из $\{0,1\}^n$ и запросить $f\left(x^0\right)$;
for $t = 1, 2, 3, \ldots$ *до окончания работы* **do**
  В зависимости от $f\left(x^0\right), \ldots, f\left(x^{t-1}\right)$ выбрать непредвзятый оператор порядка k p^t и k запрошенных ранее точек $x^{i_1}, \ldots x^{i_k}$;
  Выбрать x^t из $p^t\left(x^{i_1}, \ldots x^{i_k}\right)$ и запросить $f\left(x^t\right)$;
end

Теперь формально определим black-box сложность. Мы будем оценивать исключительно число запросов к оракулу, считая все остальные вычисления бесплатными. Время работы $T_{A,f}$ вероятностного алгоритма A для функции f — это ожидаемое число запросов до момента нахождения точки оптимума.

Определение 2 (Black-box сложность)**.** *Сложность класса функций* $\mathcal{F}$ *относительно класса алгоритмов* $\mathcal{A}$ *определяется как*

$$T_{\mathcal{A}}(\mathcal{F}) = \min_{A \in \mathcal{A}} \max_{f \in \mathcal{F}} T_{A,f}.$$

1.2. Класс функций Jump

Определим формально классы функций $\textsc{OneMax}_n$ — число единиц в числе, и $\textsc{Jump}_{n,\ell}$ — те же функции, но значения, меньшие ℓ и симметричные им заменены на нули.

Определение 3. *Для всех* $n \in \mathbb{N}$ *и* $z \in \{0,1\}^n$*, определим функцию* $\textsc{OneMax}_{n,z} : \{0,1\}^n \to \mathbb{N}$ *как*

$$\textsc{OneMax}_{n,z}(x) = \left|\{j \in [n] \mid x_j = z_j\}\right|_1.$$

Также зададим класс функций $\textsc{OneMax}_n$ *как*

$$\textsc{OneMax}_n = \left\{\textsc{OneMax}_{n,z} \mid z \in \{0,1\}^n\right\}.$$

Определение 4. *Для* $n \in \mathbb{N}$*,* $\ell < \lfloor n/2 \rfloor$ *и* $z \in \{0,1\}^n$*, зададим функцию* $\textsc{Jump}_{n,\ell,z} : \{0,1\}^n \to \mathbb{N}$ *следующим образом:*

$$\textsc{Jump}_{n,\ell,z}(x) = \begin{cases} n, & \text{при } \textsc{OneMax}_{n,z}(x) = n; \\ \textsc{OneMax}_{n,z}(x), & \text{при } \ell < \textsc{OneMax}_{n,z}(x) < n - \ell; \\ 0, & \text{иначе.} \end{cases}$$

Класс функций $\textsc{Jump}_{n,\ell}$ *определяется как*

$$\textsc{Jump}_{n,\ell} = \left\{\textsc{Jump}_{n,\ell,z} \mid z \in \{0,1\}^n\right\}.$$

Чтобы понять причину таких определений, достаточно вспомнить, что модель без ограничений рассматривает классы функций, а не отдельные функции, поскольку для оптимизации одной функции достаточно просто вернуть оптимум на первом шаге.

В непредвзятой же модели любой алгоритм работает одинаково на всех функциях из любого из определенных выше классов всилу инвариантности по $\oplus$, и оптимизация одной конкретной функции, как, например, $\textsc{Jump}_{n,\ell,(1,1,\ldots,1)}$, имеет ровно ту же сложность, что и оптимизация всего класса.

Глава 2. Black-Box оптимизация функций Jump в неограниченной модели

В этой главе мы приведем алгоритмы, дающие верхние оценки на сложность $\textsc{Jump}_{n,\ell}$ при различных параметрах, и объединим все результаты в одну общую верхнюю оценку, покрывающую все случаи.

2.1. Небольшие значения параметра Jump

В первой теореме мы докажем, что если ℓ достаточно невелико, $\textsc{Jump}_{n,\ell}$ может быть оптимизирован при помощи простого алгоритма, использующего те же идеи, что и известный алгоритм для $\textsc{OneMax}_n$.

Теорема 1. *Для $\ell < n/2 - \sqrt{n}\log n$, black-box сложность $\textsc{Jump}_{n,\ell}$ в неограниченной модели не превышает $(1 + o(1))\frac{2n}{\log n}$, где $o(1)$ измеряется относительно n.*

Эта теорема очень похожа на теорему 6 из [DJK+11], дающую верхнюю оценку на сложность $\textsc{OneMax}_n$. Мы будем использовать тот же алгоритм, и единственная разница будет состоять в том, что теперь некоторые запросы будут возвращать нули вместо соответствующих значений OneMax. В доказательстве мы покажем, что информации от ненулевых запросов достаточно, чтобы восстановить ответ, но прежде сформулируем два вспомогательных утверждения.

Следующее неравенство используется в [DJK+11] для доказательства верхней границы на сложность OneMax.

Утверждение 1. *Для достаточно большого n, $t \geq \left(1 + \frac{4 \log\log n}{\log n}\right) \frac{2n}{\log n}$, и четного $d \in \{2, \ldots, n\}$, выполнено неравенство $\binom{n}{d} \left(\binom{d}{d/2} 2^{-d}\right)^t \leq 2^{-3t/4}$*

Доказательство. Это утверждение сформулировано и доказано как утверждение 8 в [DJK+11]. □

Утверждение 2. *Для достаточно большого n, $\ell < n/2 - \sqrt{n} \log n$, $z \in \{0,1\}^n$ и x, выбранного случайно из $\{0,1\}^n$, вероятность того, что $\textsc{Jump}_{n,\ell,z}(x)$ равно нулю, не превышает $2 \exp\left(-2 (\log n)^2\right)$.*

Доказательство. Значение OneMax для случайного $x \in \{0,1\}^n$ имеет биномиальное распределение с параметрами n и $p = 1/2$. Из неравенства Хёфдинга известно, что для $k \leq np$, функция распределения для биномиального распределения $F_{n,p}(k)$ ограничена сверху выражением $\exp\left(-2\frac{(np-k)^2}{n}\right)$. Как следствие, вероятность того, что $\textsc{Jump}_{n,\ell,z}(x)$ равно нулю, не превышает $2F_{n,1/2}(\ell) \leq 2e^{-2\frac{(n/2-\ell)^2}{n}} \leq 2e^{-2\frac{\left(\sqrt{n}\log n\right)^2}{n}} = 2e^{-2(\log n)^2}$. □

Теорема 2. *Пусть n достаточно велико и $\ell < n/2 - \sqrt{n} \log n$. Пусть также $z \in \{0,1\}^n$, а X — множество из $t \geq \left(1 + \frac{4 \log\log n}{\log n}\right) \frac{2n}{\log n}$ элементов $\{0,1\}^n$, выбранных случайно с равномерным распределением и взаимно независимых. Тогда вероятность того, что существует $y \in \{0,1\}^n$, такое, что $y \neq z$ и $\textsc{Jump}_{n,\ell,z}(x) = \textsc{Jump}_{n,\ell,y}(x)$ для всех $x \in X$, не превышает $2^{-t/4}$.*

Доказательство. Пусть n достаточно велико, $z \in \{0,1\}^n$, а X — множество из t равномерно случайно и независимо выбранных элементов $\{0,1\}^n$. Для $d \in [n]$, обозначим за A_d множество точек, находящихся на расстоянии Хэмминга d от z. Будем говорить, что точка $y \in \{0,1\}^n$ согласована с $x \in X$, если $\textsc{Jump}_{n,\ell,z}(x) = \textsc{Jump}_{n,\ell,y}(x)$. Такое равенство означает, что либо $\textsc{Jump}_{n,\ell,z}(x) = 0$, либо $\textsc{OneMax}_{n,y}(x) = \textsc{OneMax}_{n,z}(x)$. Вероятность первого не превышает $2 \exp\left(-2 (\log n)^2\right)$ по утверждению 2, второе же выполняется тогда и только тогда, когда x и y (а также x и z) различаются в точности в половине из d битов, в которых отличаются y и z.

Таким образом, вероятность того, что фиксированное y согласовано со равномерно случайно выбранным x ограничена сверху $2 \exp\left(-2 (\log n)^2\right)$ для нечетного d и $2 \exp\left(-2 (\log n)^2\right) + \binom{d}{d/2} 2^{-d}$ для четного d.

Обозначим за p вероятность того, что существует $y \in \{0,1\}^n \setminus \{z\}$, такое, что y согласовано со всеми $x \in X$. Тогда

$$p = Pr\left(\bigcup_{y\in\{0,1\}^n\setminus\{z\}} \bigcap_{x\in X} \text{“y согласовано с x”}\right).$$

$$p \leq \sum_{y\in\{0,1\}^n\setminus\{z\}} Pr\left(\bigcap_{x\in X} \text{“y согласовано с x”}\right).$$

Поскольку события “y согласовано с x” взаимно независимы для фиксированного y и $x \in X$,

$$p \leq \sum_{d=1}^{n} \sum_{y\in A_d} \prod_{x\in X} Pr\left(\text{“y согласовано с”}\right).$$

Применяя оценки на вероятности, полученные выше, получаем

$$p \leq \sum_{d \text{ even}} \sum_{y\in A_d} \binom{n}{d} \left(2e^{-2(\log n)^2} + \binom{d}{d/2}2^{-d}\right)^t + \sum_{d \text{ odd}} \sum_{y\in A_d} \binom{n}{d} \left(2e^{-2(\log n)^2}\right)^t.$$

Для достаточно большого n,

$$2e^{-2(\log n)^2} \leq \left(2^{1/4} - 1\right)\binom{d}{d/2}2^{-d},$$

поэтому можно написать, что

$$p \leq \sum_{d \text{ even}} \sum_{y\in A_d} \binom{n}{d} \left(2^{1/4}\binom{d}{d/2}2^{-d}\right)^t + 2^n\left(2e^{-2(\log n)^2}\right)^t.$$

Применяя утверждение 1, получаем

$$p \leq n2^{t/4}2^{-3t/4} + 2^{n+t}e^{-2t(\log n)^2},$$

что меньше $2^{-t/4}$ для достаточно большого n.

□

Доказательство теоремы 1. Алгоритм 3 — тот же алгоритм, что используется в [DJK+11] для доказательства оценки на сложность OneMax.

Мы равномерно случайным образом выбираем множество запросов и проверяем, верно ли, что существует ровно один оптимум, согласующийся со всей информацией о значениях функции. Сложность одной итерации алгоритма — t операций, а вероятность неудачи не превышает $2^{-t/4}$ по теореме 2. Следовательно, сложность алгоритма 3 ограничена сверху как $\frac{t}{1-2^{-t/4}} = (1 + o(1)) \frac{2n}{\log n}$.

Алгоритм 3: Black-box алгоритм для $\textsc{Jump}_{n,\ell}$, где $\ell < \frac{n}{2} - \sqrt{n} \log n$, неограниченная модель

Input: $f \in \textsc{Jump}_{n,\ell}$
$t \leftarrow \left(1 + \frac{4 \log \log n}{\log n}\right) \frac{2n}{\log n}$;
repeat
  for $i = 1$ **to** t **do**
    $x_i = \texttt{uniform}()$;
    $f_i = f(x_i)$;
  end
  $w = \texttt{chooseConsistentSelected}\left(\{x_i, f_i\}_{i=1}^{t}\right)$;
until w — *оптимум*;
return w;

□

2.2. Оптимизация Jump при помощи сведения к меньшим задачам

Для больших значений ℓ простой выбор случайных запросов не даст хороших оценок на сложность, поскольку шанс получить точку со значением функции, равным нулю, можкт быть очень большим, и такой запрос не раскрывает много информации об ответе. В следующей теореме мы предложим другой способ оптимизировать функции класса Jump путем задействования любого из известных алгоритмов для OneMax.

Теорема 3. *Для $\ell < \left\lfloor \frac{n}{2} \right\rfloor - 1$, black-box сложность $\textsc{Jump}_{n,\ell}$ в неограниченной модели не превышает $(2 + o(1)) \frac{n}{\log(n-2\ell)}$ где $o(1)$ рассматривается при $(n - 2\ell) \to \infty$.*

Доказательство. Для доказательства этой верхней оценки, сведем оптимизацию $\textsc{Jump}_{n,\ell}$ к оптимизации $\textsc{OneMax}_k$, где $k = \left\lfloor \frac{n}{2} \right\rfloor - \ell - 1$. Это позволит использовать результат [DJK+11], дающий верхнюю оценку на сложность $\textsc{OneMax}_k$, для получения оценки на сложность $\textsc{Jump}_{n,\ell}$.

Алгоритм 4: Black-box алгоритм для $\textsc{Jump}_{n,\ell}$, $\ell < \left\lfloor \frac{n}{2} \right\rfloor - 1$, неограниченная модель

Input: $f \in \textsc{Jump}_{n,\ell}$
repeat
  $x \leftarrow \texttt{uniform}()$;
until $f(x) = \left\lfloor \frac{n}{2} \right\rfloor$;
$k \leftarrow \left\lfloor \frac{n}{2} \right\rfloor - \ell - 1$;
$\tau \leftarrow \lceil n/k \rceil$;
for $i = 1$ **to** τ **do**
  $x = \alpha\beta\gamma$, $|\alpha| = (i-1)\,k$, $|\beta| = \min\{k, n - |\alpha|\}$
  $\alpha\omega_i\gamma \leftarrow \texttt{optimizeSelected}\left(\alpha\beta\gamma, \alpha\overline{\beta}\gamma\right)$;
end
return $\omega_1\omega_2\ldots\omega_\tau$

Алгоритм 5: Функция `optimizeSelected`, использующаяся в алгоритме 4

Input: Две двоичные строки $\alpha\beta\gamma, \alpha\overline{\beta}\gamma$ длины n, различающиеся в $s = |\beta|$ позициях
Output: Строка $\alpha\omega\gamma$, совпадающая с оптимумом в части ω
A — генератор запросов любого black-box алгоритма для $\textsc{OneMax}_s$;
$\Delta \leftarrow \frac{1}{2}\left(f(\alpha\beta\gamma) + f\left(\alpha\overline{\beta}\gamma\right) - s\right)$;
$l \leftarrow 0$;
repeat
  $x_{l+1} \leftarrow A\left(\{x_i, f_i\}_{i=1}^{l}\right)$;
  $f_{l+1} \leftarrow f(\alpha x_{l+1}\gamma) - \Delta$;
  $l \leftarrow l + 1$;
until $f_l = s$;
return $\alpha x_l \gamma$

Алгоритм 4 начинает работу с того, что находит точку x со значением функции, в точности равным $\left\lfloor \frac{n}{2} \right\rfloor$. Вероятность того, что функция равна $\left\lfloor \frac{n}{2} \right\rfloor$ в случайной точке, равна $2^{-n}\binom{n}{\lfloor n/2 \rfloor}$, что по формуле Стирлинга есть $\Theta\left(\frac{1}{\sqrt{n}}\right)$, поэтому сложность первого цикла алгоритма равна $\Theta(\sqrt{n})$.

После нахождения такого x, мы можем получить значение OneMax в любой точке, различающейся с x в не более, чем k позициях. Это позволяет решать задачу, разбив все битовые позиции на множества размера не более k каждое и оптимизируя каждое множество отдельно: если ограничиться одним таким множеством, можно получать о нем полную информацию и оптимизировать его при помощи алгоритма для OneMax. Это и реализовано во втором цикле алгоритма: на каждой итерации мы выбираем не более k позиций (ровно k на всех итерациях, кроме последней, и не более k на последней) и вызываем `optimizeSelected` (алгоритм 5) для оптимизации выбранных позиций.

Сложность второй части алгоритма, таким образом, можно явно записать как сложность $\lfloor n/k \rfloor$ вызовов `optimizeSelected` с k выбранными битами и, возможно, еще один вызов с $(n \mod k)$ выбранными битами.

Функция `optimizeSelected` для s выбранных битов может быть реализована при помощи любого алгоритма A для OneMax_s, и использует ровно на две операции больше, чем A. По теореме 6 из [DJK+11], A может быть выбран таким образом, что ожидаемое число операций будет равно $\frac{(2+o(1))s}{\log s}$, что приводит к оценке $\frac{(2+o(1))s}{\log s} + 2 = \frac{(2+o(1))s}{\log s}$ на сложность `optimizeSelected`.

Таким образом, если $n = kq + r$, где $0 < r \leq k$, суммарная сложность всех `optimizeSelected` не превышает

$$q\frac{(2+o_k(1))\,k}{\log k} + \frac{(2+o_r(1))\,r}{\log r} = \frac{2+o_k(1)}{\log k}\left(qk + r\frac{(2+o_r(1))\log k}{(2+o_k(1))\log r}\right) =$$
$$= \frac{2+o_k(1)}{\log k}(qk+r)\left(1 + \frac{r\,(1+o_r(1))\,(\log k - \log r)}{(qk+r)\log r}\right) =$$
$$= \frac{2+o_k(1)}{\log k}(qk+r)\left(1 + \frac{(1+o_r(1))\log(k/r)}{(1+qk/r)\log r}\right) = \frac{(2+o_k(1))\,n}{\log k}.$$

Общую сложность алгоритма 4 теперь можно записать как

$$O\left(\sqrt{n}\right) + \frac{(2+o_k(1))\,n}{\log k} = \frac{(2+o_{n-2\ell}(1))\,n}{\log(n-2\ell)},$$

что завершает доказательство. □

В предыдущей теореме мы успешно решили Jump путем сведения к решению OneMax меньшего размера. Следующая теорема

продемонстрирует, что вместо OneMax можно свести задачу к случаю Jump, для которого у нас уже есть решение из теоремы 1. Результатом такого подхода будет более низкий множитель в верхней оценке сложности, сильно приближающий ее к нижней оценке, которая будет дана позднее.

Теорема 4. *Для* $\frac{n}{2} - \sqrt{n}\log n \leq \ell < \left\lfloor \frac{n}{2} \right\rfloor - 1$, *black-box сложность* $\textsc{Jump}_{n,\ell}$ *в неограниченной модели не более* $(1 + o(1)) \frac{n}{\log(n-2\ell)}$, *где* $o(1)$ *рассматривается при* $(n - 2\ell) \to \infty$.

Доказательство. Пусть $k = \left\lfloor \frac{n}{2} \right\rfloor - \ell - 1$. Так же, как и в доказательстве предыдущей теоремы, мы будем оптимизировать $\textsc{Jump}_{n,\ell}$ путем сведения к более простой задаче, но в этот раз вместо $\textsc{OneMax}_k$ мы будем использовать $\textsc{Jump}_{s,\frac{s}{2}-k-1}$ с $\sqrt{s}\log s < k$.

Алгоритм 6: Black-box алгоритм для $\textsc{Jump}_{n,\ell}$, где $\frac{n}{2} - \sqrt{n}\log n \leq \ell < \left\lfloor \frac{n}{2} \right\rfloor - 1$, неограниченная модель

Input: $f \in \textsc{Jump}_{n,\ell}$
$k \leftarrow \left\lfloor \frac{n}{2} \right\rfloor - \ell - 1$;
$s \leftarrow \max\{w \mid \sqrt{w}\log w < k; w \text{ is even}\}$;
$\tau \leftarrow \lceil n/s \rceil$;
repeat
  $x \leftarrow \texttt{uniform}()$;
until $f(x) = \left\lfloor \frac{n}{2} \right\rfloor$;
$B \leftarrow [1, n]$;
for $i = 1$ **to** $\tau - 1$ **do**
  $b_i \leftarrow \texttt{selectBits}_{s,B}(x)$;
  $B \leftarrow B \setminus b_i$;
end
$b_\tau \leftarrow B$;
$\omega_0 \leftarrow x$;
for $i = 1$ **to** τ **do**
  $\alpha_i \leftarrow \texttt{optimizeSelected}_{b_i}(x)$;
  $\omega_i \leftarrow \texttt{setAtPositions}_{b_i}(\omega_{i-1}, \alpha_i)$;
end
return ω_τ;

Алгоритм, который мы будем использовать для доказательства этой верхней оценки, приведен как алгоритм 6. Он начинает работу с выбора максимального четного s, которое все еще достаточно мало, чтобы можно было применить теорему 1 к сложности $\textsc{Jump}_{s,\frac{s}{2}-k-1}$.

Алгоритм 7: Функция `selectBits`, используемая в алгоритме 6

Input: Двоичная строка x, четное s и множество битовых позиций B, такое, что $|B| \geq s$, x совпадает с ответом в точности в $\frac{|B|}{2}$ позициях из множества B и $\left\lfloor \frac{|x|-|B|}{2} \right\rfloor$ позициях не из B.

Output: Множество позиций $b \subset B$, такое, что $|b| = s$, и x совпадает с ответом в ровно $s/2$ позициях из множества b.

repeat

  $b \leftarrow \texttt{chooseSubsetRandomlyUniformly}\,(B, s)$;

until $f\left(\texttt{flipBits}_b\left(x\right)\right) = f\left(x\right)$;

return b

Алгоритм 8: Функция `optimizeSelected`, используемая в алгоритме 6

Input: Двоичная строка x и множество позиций b, такое, что $|b| \leq s$, и x совпадает с ответом в ровно $\left\lfloor \frac{|x|-|b|}{2} \right\rfloor$ позициях среди тех, что не принадлежат b.

Output: Двоичная строка ω, совпадающая с ответом в выбранных позициях

Не умаляя общности, можно считать, что b — это первые несколько позиций, $x = \beta\gamma$;

$k \leftarrow \left\lfloor \frac{n}{2} \right\rfloor - \ell - 1$;

A — генератор запросов black-box алгоритма для $\textsc{Jump}_{|\beta|,\left\lfloor \frac{|\beta|}{2} \right\rfloor - k - 1}$;

$\Delta \leftarrow \left\lfloor \frac{|\gamma|}{2} \right\rfloor$;

$l \leftarrow 0$;

repeat

  $x_{l+1} \leftarrow A\left(\left\{x_i, f_i\right\}_{i=1}^{l}\right)$;

  $f_{l+1} \leftarrow \begin{cases} 0, & \text{при } f\left(x_{l+1}\gamma\right) = 0 \\ f\left(x_{l+1}\gamma\right) - \Delta, & \text{иначе} \end{cases}$;

  $l \leftarrow l + 1$;

until x_l — *оптимум*;

return $x_l\gamma$;

После этого, так же, как и в алгоритме 4, мы найдем двоичную строку x с ровно $\left\lfloor \frac{n}{2} \right\rfloor$ верными битами путем случайных запросов. Сложность этой части, как и в предыдущем алгоритме, равна $\Theta\left(\sqrt{n}\right)$.

После нахождения подходящего x, мы делим все позиции на множества размера s каждое (кроме, быть может, одного) таким образом, что в каждом множестве ровно половина битов совпадают с ответом. Это делает второй цикл алгоритма при помощи функции `selectBits`, описанной подробно как алгоритм 7. Эта функция принимает x и множество B из не менее, чем s позиций, обладающее тем свойством, что одна половина битов в множестве совпадают с ответом, а оставшиеся — нет, и находит подмножество B размера ровно s, обладающее тем же свойством. Чтобы реализовать такую функцию `selectBits`, будем просто выбирать случайные подмножества из s позиций и проверять, выполняется ли для них нужное условие. Легко видеть, что условие выполнено в том и только в том случае, когда замена всех этих битов на противоположные не изменяет значение функции Jump. Если положить размер множества B за m, то число таких подмножеств можно записать как $\binom{\lfloor m \rfloor/2}{s/2}\binom{\lceil m/2 \rceil}{s/2}$, а общее число подмножеств размера s равно $\binom{m}{s}$. Таким образом, вероятность успеха одной итерации можно вычислить как

$$\binom{\lfloor m/2 \rfloor}{s/2}\binom{\lceil m/2 \rceil}{s/2}\binom{m}{s}^{-1} =$$

$$= \frac{\lfloor m/2 \rfloor!\,\lceil m/2 \rceil!\, s!\,(m-s)!}{\lfloor (m-s)/2 \rfloor!\,\lceil (m-s)/2 \rceil!\,(s/2)!^2} = \binom{s}{s/2}\binom{m-s}{\lfloor (m-s)/2 \rfloor}\binom{m}{\lfloor m/2 \rfloor}^{-1} =$$

$$= \Theta\left(\frac{\left(2^s/\sqrt{s}\right)\left(2^{m-s}/\sqrt{m-s}\right)}{2^m/\sqrt{m}}\right) = \Theta\left(\frac{\sqrt{m}}{\sqrt{s}\sqrt{m-s}}\right) = \Omega\left(\frac{1}{\sqrt{s}}\right).$$

Это дает оценку сложности в $O\left(\sqrt{s}\right)$ на функцию `selectBits` и $O\left(\frac{n}{\sqrt{s}}\right)$ на всю вторую часть алгоритма.

Наконец, в третьем цикле алгоритма мы берем выранные множества битовых позиций и оптимизируем каждое в отдельности. Это реализовано в функции `optimizeSelected` (алгоритм 8) с использованием того факта, что оптимизация одного такого множества эквивалента оптимизации функции Jump меньшего размера. В самом деле, если изменять биты только в выбранном множестве, зафиксировав остальные, получаемые значения функции будут равны значениям Jump для этого множества, сдвинутым на константу, за исключением двух случаев: нули остаются нулями, и

в случае, когда меньший Jump вернул бы, что точка является ответом, мы также получим ноль. Второй случай — это единственный, когда мы "теряем" информацию о значении функции, но он, как можно заметить, не влияет ни на корректность, ни на сложность алгоритма 3, поскольку изменение вероятности успеха для `chooseConsistentSelected` слишком незначительно, а определить наличие уникального оптимума возможно без явного вычисления функции для него. Следовательно, сложность оптимизации одного множества b можно ограничить, использовав теорему 1, и она не превышает $\left(1 + o_{|b|}\left(1\right)\right) \frac{2|b|}{\log|b|}$.

После того, как мы нашли ответ для каждого множества, мы просто объединяем результаты, формируя тем самым ответ для $\textsc{Jump}_{n,\ell}$.

Сложность всего алгоритма для $n = qs+r$ $(0 < r \leq s)$ можно записать как

$$O\left(\sqrt{n}\right) + O\left(\frac{n}{\sqrt{s}}\right) + q\left(2 + o_s\left(1\right)\right)\frac{s}{\log s} + \left(2 + o_r\left(1\right)\right)\frac{r}{\log r}.$$

Заметим, что это выражение очень похоже на то, что было в доказательстве теоремы 3, и аналогичными рассуждениями может быть преобразовано в $\frac{(2+o_s(1))n}{\log s}$. Осталось лишь заметить, что из выбора s выполняется равенство $\log s = (2 + o_k\left(1\right))\log k$, что приводит к окончательному результату в $\frac{(1+o_{n-2\ell}(1))n}{\log(n-2\ell)}$, завершая тем самым доказательство.

□

2.3. Оптимизация Jump с крайним значением параметра

Алгоритмы, которые мы использовали для доказательства предыдущих верхних оценок, опираются на то, что $k = \left\lfloor \frac{n}{2} \right\rfloor - 1$ не равно нулю, поэтому остается непокрытым случай $\ell = \left\lfloor \frac{n}{2} \right\rfloor - 1$, в котором единственные точки с ненулевым значением функции — это непосредственно ответ и точки с ровно $\left\lfloor \frac{n}{2} \right\rfloor$ или $\left\lceil \frac{n}{2} \right\rceil$ правильными битами. В дальнейшем мы будем называть такие функции Jump как экстремальный Jump.

В следующей теореме мы приведем другой алгоритм для поиска оптимума в случае экстремального Jump, который, как мы докажем в

дальнейшем, имеет наименьшую возможную сложнсть за исключением членов меньшего порядка.

Теорема 5. *Для* $\ell = \left\lfloor \frac{n}{2} \right\rfloor - 1$, *black-box сложность* $\textsc{Jump}_{n,\ell}$ *в неограниченной модели не превышает* $n + O\left(\sqrt{n}\right)$.

Доказательство. Наш алгоритм будет основан на проверке индивидуальных битов. Допустим, что у нас есть точка x, такая, что $f(x) = \left\lfloor \frac{n}{2} \right\rfloor$. Если мы заменим ровно два бита x на противоположные, значение функции останется прежним в том и только в том случае, когда один бит был верным, а второй нет. Таким образом, за один запрос мы можем узнать исключающее или любых двух битов, и, применив это к парам из первого бита и каждого из оставшихся, мы можем найти оптимум или его отрицание.

Алгоритм 9: Black-box алгоритм для экстремального Jump, неограниченная модель

Input: $f \in \textsc{Jump}_{n,\ell}$, где $\ell = \left\lfloor \frac{n}{2} \right\rfloor - 1$
repeat
  $x \leftarrow \texttt{uniform}()$;
until $f(x) = \left\lfloor \frac{n}{2} \right\rfloor$;
for $i = 1$ **to** $n - 1$ **do**
  $y^i \leftarrow x \bigoplus 10^{i-1}10^{n-1-i}$;
  if $f(y^i) = f(x)$ **then**
    $b_i \leftarrow 0$;
  else
    $b_i \leftarrow 1$;
  end
end
$u \leftarrow x \bigoplus 0b_1b_2b_3\ldots b_{n-1}$;
$v \leftarrow x \bigoplus 1\overline{b_1b_2b_3\ldots b_{n-1}}$;
if $f(u) = n$ **then**
  return u;
else
  return v;
end

Алгоритм 9 реализует эту идею, и его сложность может быть оценена как $\Theta\left(\sqrt{n}\right)$ для перой части и $n+1$ для второй, что в сумме дает $n+O\left(\sqrt{n}\right)$. □

Объединяя результаты теорем 1, 4 и 5, мы получаем верхнюю оценку на black-box сложность класса Jump в общем случае.

Теорема 6. *Для всех* $\ell \leq \left\lfloor \frac{n}{2} \right\rfloor - 1$, *black-box сложность класса* $\textsc{Jump}_{n,\ell}$ *в неограниченной модели не более* $\frac{(1+o(1))n}{\log \min\{(n-2\ell),\sqrt{n}\}}$, *где* $o(1)$ *рассматривается при* $(n-2\ell) \to \infty$.

Доказательство. Это результат объединения верхних оценок из теорем 1, 4 и 5 на сложность $\textsc{Jump}_{n,\ell}$ в зависимости от значения ℓ. □

Глава 3. Оптимизация в непредвзятой модели

Алгоритм, который мы использовали для оценки сверху при помощи сведения к OneMax (алгоритм 4), не опирается сильно на неограниченность модели. Таким образом, он может быть реализован в непредвзятой модели, что дает следующий результат.

Теорема 7. *Для* $\ell < \left\lfloor \frac{n}{2} \right\rfloor - 1$ *и* $m \geq 3$, *black-box сложность* $\textsc{Jump}_{n,\ell}$ *в непредвзятой модели порядка* m *равна* $O\left(\frac{n}{\min\{\log(n-2\ell), m\}}\right)$.

Доказательство. Алгоритм 10 реализует идею алгоритма 4 в непредвзятой модели порядка m для $m \geq 3$.

Отличия будут состоять в том, что теперь выбор битов нужно реализовать при помощи непредвзятых операторов порядка 3, а функция `optimizeSelected` будет использовать алгоритм для OneMax порядка $(m-1)$.

Первым делом, мы находим точку x со значением функции, в точности равным $\lfloor n/2 \rfloor$, путем запросов из случайно равномерно выбранных точек. Очевидно, такие запросы допустимы в нашей модели, и сложнсть этой части $\Theta(\sqrt{n})$.

После этого, алгоритм для неограниченной модели разбивал все позиции на множества размера не более $k = \left\lfloor \frac{n}{2} \right\rfloor - \ell - 1$, и оптимизировал их по отдельности. Чтобы реализовать выбор битов в непредвзятой модели, будем поддерживать две битовые строки в дополнение к x после i итераций алгоритма:

- z_i различается с x в обработанных ik позициях и совпадает во всех остальных;

- u_i совпадает с ответом в обработанных позициях (то есть в позициях, в которых различаются x и z_i).

На i итерации, мы выбираем $\Delta = \min\{k, n-(i-1)\,k\}$ из необработанных позиций при помощи бинарного непредвзятого оператора `flipWhereEqual`. После этого, мы оптимизируем выбранные биты, используя функцию `optimizeSelected`, приведенную как алгоритм 11. Эта функция берет две строки, различающиеся в не более, чем k позициях, и возвращает строку, совпадающую с ответом в этих позициях.

Алгоритм 11 работает так же, как и алгоритм для аналогичной функции в неограниченной модели: мы используем алгоритм для OneMax, и в этом случае на каждом шаге он дает вероятностное распределение, зависящее от $m-1$ из предыдущих запросов непредвзятым образом.

Чтобы применить алгоритм для OneMax к нашей задаче, будем преобразовывать запросы x_i для OneMax_s в αx_i, где α — фиксированная часть, и в дополнение к этому будем поддерживать отрицания запросов $\alpha\overline{x_i}$. Это позволит нам преобразовать непредвзятое распределение порядка $(m-1)$ на выбранных битах в распределение порядка m: добавив отрицание одного из запросов, мы получим возможность отделить выбранные позиции от невыбранных (выбранные позиции — это те и только те, в которых все запросы совпадают), поэтому распределение, заданное в алгоритме, действительно будет непредвзятым порядка m. После того, как мы сгенерировали очередной запрос, мы можем получить его отрицание при помощи оператора порядка 3, тем самым поддержав необходимые инварианты и завершив шаг алгоритма.

Получив результат `optimizeSelected`, мы просто пересчитываем z_i и u_i при помощи непредвзятых операторов порядка 3 `update1` и `update2`, и завершаем итерацию.

Анализ сложности будет таким же, как и в неограниченном случае, за исключением того, что теперь используется другой алгоритм для OneMax. Результатом будет $O\left(\sqrt{n}\right)$ для первой части и суммарная сложность $\lceil n/k \rceil$ вызовов OneMax размера не более k для второй части, где $k = \left\lfloor \frac{n}{2} \right\rfloor - \ell - 1$.

Из результатов [DW], сложность OneMax_k в непредвзятой модели порядка $(m-1)$ равна $O\left(\frac{k}{\min\{\log k, m-1\}}\right)$, что приводит к общей сложнсти

алгоритма в

$$\lceil n/k \rceil \, O\left(\frac{k}{\min\{\log k, m-1\}}\right) = O\left(\frac{n}{\min\{\log k, m-1\}}\right) = \\ = O\left(\frac{n}{\min\{\log(n-2\ell), m\}}\right).$$

Алгоритм 10: Оптимизация JUMP при помощи непредвзятых операторов порядка m, $m \geq 3$

Input: $f \in \text{JUMP}_{n,\ell}$
repeat
  $x \leftarrow \texttt{uniform}()$;
until $f(x) = \left\lfloor \frac{n}{2} \right\rfloor$;
$u^{(0)} \leftarrow x, z^{(0)} \leftarrow x$;
$k \leftarrow \left\lfloor \frac{n}{2} \right\rfloor - \ell - 1$;
$\tau \leftarrow \lceil n/k \rceil$;
for $i = 1$ **to** τ **do**
  $s = \min\{k, n - (i-1)\,k\}$;
  $y \leftarrow \texttt{flipWhereEqual}_s\left(x, z^{(i-1)}\right)$;
  Положим $x = \alpha\beta\gamma$, $z^{(i-1)} = \overline{\alpha}\beta\gamma$, $y = \alpha\overline{\beta}\gamma$;
  $\alpha\omega_i\gamma \leftarrow \texttt{optimizeSelected}\left(\alpha\beta\gamma, \alpha\overline{\beta}\gamma\right)$;
  Инвариант: $u^{(i-1)} = \omega\beta\gamma$ и все биты ω совпадают с ответом;
  $\overline{\alpha\beta}\gamma \leftarrow \texttt{update1}\left(\alpha\beta\gamma, \overline{\alpha}\beta\gamma, \alpha\overline{\beta}\gamma\right)$;
  $\omega\omega_i\gamma = \texttt{update2}\left(\omega\beta\gamma, \alpha\beta\gamma, \alpha\omega_i\gamma, \right)$;
  $u^{(i)} = \omega\omega_i\gamma$ и $z^{(i)} = \overline{\alpha\beta}\gamma$;
end
return $u^{(\tau)}$

□

Алгоритм 11: Функция `optimizeSelected`, используемая в алгоритме 10

Input: Две битовые строки $\alpha\beta, \alpha\overline{\beta}$ длины n, различающиеся в $s = |\beta|$ битах

Output: Строка $\alpha\omega$ совпадающая с оптимумом в секции ω

A — генератор запросов алгоритма для $\textsc{OneMax}_s$ в модели порядка $m-1$;

$\Delta \leftarrow \frac{1}{2}\left(f(\alpha\beta) + f\left(\alpha\overline{\beta}\right) - s\right)$;

$l \leftarrow 0$;

repeat

$D_{l+1}\left(\cdot \mid \{x_{i_k}\}_{k=1}^{m-1}\right) \leftarrow A\left(\{f_i\}_{i=1}^{l}\right)$;

$D'_{l+1}\left(y \mid \{\alpha x_{i_k}\}_{k=1}^{m-1}, \alpha\overline{x_{i_0}}\right) \leftarrow \begin{cases} D_{l+1}(\psi), & \text{при } y = \alpha\psi \\ 0, & y \text{ не вида } \alpha\psi \end{cases}$;

$\alpha x_{l+1} \leftarrow \texttt{sample}\left(D'_{l+1}\right)$;

$\alpha\overline{x_{l+1}} \leftarrow \texttt{update}\left(\alpha x_{l+1}, \alpha\beta, \alpha\overline{\beta}\right)$;

$f_{l+1} \leftarrow f(\alpha x_{l+1}) - \Delta$;

$l \leftarrow l + 1$;

until $f_l = s$;

return αx_l

Глава 4. Нижние оценки

Число различных значений, которые может принимать функция из $\textsc{Jump}_{n,\ell}$, равно $n - 2\ell + 1$, а возможных оптимумов 2^n. Это мгновенно дает нижнюю оценку в $\left\lceil \frac{n}{\log(n-2\ell)} \right\rceil - 1$ из информационно-теоретических соображений, приведенных в теореме 2 [DJW].

Теорема 8. *Для любого $\ell < \lfloor n/2 \rfloor$, black-box сложность $\textsc{Jump}_{n,\ell}$ в неограниченной модели ограничена снизу как $\left\lceil \frac{n}{\log(n-2\ell+1)} \right\rceil - 1$.*

Доказательство. Это напрямую следует из теоремы 2 в [DJW]. □

Множитель в верхних оценках из теорем 6 и 5 равен $1 + o(1)$ (где $o(1)$ при условии $n - 2\ell \to \infty$) в неэкстремальном случае (и при достаточно большом ℓ), и $1 + o_n(1)$ в экстремальном случае.

Применив только что полученную нижнюю оценку к случаю экстремального $\textsc{Jump}$ ($\ell = \left\lfloor \frac{n}{2} \right\rfloor - 1$), мы получим $\frac{n}{\log 3}$ для четного n и $\frac{n}{\log 4}$ для нечетного n, поскольку функции принимают 3 и 4 возможных значения соответственно, но наш алгоритм использует большее число операций, равное $n + o(n)$.

Интуиция подсказывает, что различные значения функции не равноправны с точки зрения уменьшения сложности: если мы, например, получим значение n, то эта точка является ответом, но такое значение не поможет в самом процессе поиска. В данном разделе мы сформулируем эту идею в виде общей леммы и используем ее, чтобы доказать, что полученная из общих соображений нижняя граница не совсем точна, и константа в сложности алгоритма для экстремального $\textsc{Jump}$ является оптимальной.

Теорема 9. *Рассмотрим задачу оптимизации с конечным пространством поиска S, где для каждого $s \in S$ существует экземпляр задачи, для которого s является единственным ответом. Будем рассматривать пары*

из запроса и значения функции, и вычислять число элементов S, которые могут быть ответами при условии, что рассматриваемое значение является ответом на запрос. Пусть c и $k \geq 2$ — две целые константы, такие, что для любого запроса существует множество из не более k значений, такое, что для всех остальных значений вместе возможно не более c различных ответов. Тогда black-box сложность этой задачи не меньше $\left\lfloor \log_k \frac{|S|}{2c} \right\rfloor - 1$.

Доказательство. Для каждого $s \in S$ выберем экземпляр задачи $I(s)$, для которого s является уникальным ответом. По принципу минимакса Яо [Yao77], минимальное ожиданемое время оптимизации в худшем в смысле экземпляра задачи случае для вероятностного алгоритма ограничено снизу ожидаемым временем работы оптимального детерминированного алгоритма для равномерного распределения на выбранных экземплярах задачи.

Любой детерменированный алгоритм можно представить в виде дерева решений, где каждая внутренняя вершина соответствует запросу, исходящие ребра соответствуют различным ответам на запрос, и для каждого $s \in S$ существует лист, соответствующий тому, что s является ответом. Можем считать, что исходящая степень любой внутренней вершины не меньше двух, в противном же случае алгоритм может просто пропустить запрос.

Назовем *слабыми* ребра, отвечающие значениям не из k описанных выше, остальные же ребра назовем *сильными*. Также назовем вершину *слабой*, если на пути от корня до нее есть хотя бы одно слабое ребро.

Заметим, что сильным вершины образуют корневое дерево с исходящей степенью не более k, и для каждой вершины, слабой или сильной, можно определить уникального *сильного предка* как последнюю сильную вершину на пути от корня до нее (в случае, если вершина является сильной, это будет она сама).

Тогда, по свойству ребер, которые мы назвали слабыми, для каждой сильной вершины есть не более c слабых листьев, для которых она является сильным предком, и, следовательно, не более $2c$ вершин всего, поскольку исходящая степень в дереве не меньше двух.

Средняя глубина листа в дереве не меньше, чем средняя

глубина сильного предка листа. Последнее же есть средняя глубина последовательности из $|S|$ сильных вершин, в которой каждая вершина встречается не более $2c$ раз.

Если отсортировать элементы этой последовательности по глубине, то i элемент не меньше i элемента отсортированной последовательности из всех сильных вершин, где каждый элемент повторяется $2c$ раз.

Из этого следует, что средняя глубина члена такой последовательности не меньше средней глубины из $\left\lfloor \frac{|S|}{2c} \right\rfloor$ вершин с минимальной глубиной в сильной части дерева. Эти вершины формируют корневое дерево с исходящей степенью не больше k, и средняя глубина вершины в таком дереве не меньше $\left\lfloor \log_k \frac{|S|}{2c} \right\rfloor - 1$.

□

В следующих двух теоремах мы продемонстрируем, как, используя эту теорему, получить правильный множитель в выражении для сложности экстремального Jump.

Теорема 10. *Для четного n, black-box сложность* $\textsc{Jump}_{n,\frac{n}{2}-1}$ *в неограниченной модели не меньше $n-2$.*

Доказательство. Эта нижняя граница — результат прямого применения теоремы 9 к задаче: размер пространства поиска 2^n, возможными значениями могут быть $\{0, n/2, n\}$, но значение n сразу же раскрывает ответ, поэтому сложность не меньше $\left\lceil \log_2 \frac{2^n}{2} \right\rceil - 1 = n - 2$. □

Теорема 11. *Для нечетного n, black-box сложность* $\textsc{Jump}_{n,\frac{n-1}{2}-1}$ *в неограниченной модели не меньше $n-3$.*

Доказательство. В этом случае возможными значениями являются $\left\{0, \frac{n-1}{2}, \frac{n+1}{2}, n\right\}$. Одного использования доказанной теоремы недостаточно, но можно использовать дополнительное рассуждение, чтобы уменьшить число значений.

Рассмотрим оригинальную задачу с двумя небольшими модификациями:

1. Множество возможных экземпляров задачи ограничено теми, где ответ содержит четное число единиц.

2. Значения $\frac{n-1}{2}$ и $\frac{n+1}{2}$ заменены на 1.

Очевидно, первая модификация не увеличивает сложность задачи, так как она лишь снижает множество возможных экземпляров.

Но не увеличивает ее и вторая модификация: если мы получили значение 1, то можно однозначно определить, было ли это $\frac{n-1}{2}$ или $\frac{n+1}{2}$ исходя из четности числа единиц в запросе, поскольку значение OneMax для x нечетно, если x содержит четное число единиц, и четно в противном случае.

Это означает, что любой алгоритм для случая без второй модификации может быть применен и к измененной задаче.

Таким образом, black-box сложность измененной задачи не больше, чем сложность оригинальной задачи. В измененной задаче значениями могут быть только $\{0, 1, n\}$, и значение n сразу же выдает ответ.

Применяя теорему 9 к измененной задаче, мы получаем нижнюю границу в $\left\lceil \log_2 \frac{2^{n-1}}{2} \right\rceil - 1 = n - 3$. □

Заключение

Результатами данной работы являются достаточно точные оценки сложности для класса функций Jump, как верхние, так и нижние. Полученные оценки позволяют закрыть многие вопросы в отношении этой задачи, а идеи, использованные в доказательствах, могут быть использованы в будущем для исследования других классов функций.

Полученные оценки следуют той же тенденции, что и оценки для OneMax. Почти полное совпадение верхней и очень низкой нижней оценки для неограниченной модели подтверждает расхождение между сложностью в этой модели и тем, насколько на самом деле задача сложна для решения рандомизированным поиском. Результат, полученный для непредвзятой модели, хоть и ближе к истине, но все же показывает, что и она слишком сильна, и нужно наложить некие дополнительные ограничения на допустимые алгоритмы.

Среди открытых вопросов в отношении класса Jump остается небольшой зазор между верхней и нижней оценками для непредвзятой модели (что является частью общей проблемы того, что известно очень немного нижних оценок сложности более сильных, чем можно получить из соображений теории информации), а также сложность в случае унарных и бинарных операторов.

Литература

[AAD11] Anne Auger, Anne Auger, and Benjamin Doerr. *Theory of Randomized Search Heuristics: Foundations and Recent Developments.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2011.

[DJK+11] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Per Kristian Lehre, Markus Wagner, and Carola Winzen. Faster black-box algorithms through higher arity operators. In *Proc. of FOGA (Foundations of genetic algorithms)*, pages 163–172, 2011.

[DJW] Stefan Droste, Thomas Jansen, and Ingo Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory Comput. Syst.*, (4):525–544.

[DKW11] Benjamin Doerr, Timo Kötzing, and Carola Winzen. Too fast unbiased black-box algorithms. In *Proc. of GECCO (Genetic and evolutionary computation)*, pages 2043–2050, 2011.

[DW] Benjamin Doerr and Carola Winzen. Reducing the arity in unbiased black-box complexity. *CoRR*.

[Jan13] Thomas Jansen. *Analyzing Evolutionary Algorithms - The Computer Science Perspective.* Natural Computing Series. Springer, 2013.

[LW10] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 1441–1448, New York, NY, USA, 2010. ACM.

[NW10] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational*

Complexity. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[Yao77] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *FOCS*, pages 222–227. IEEE Computer Society, 1977.