Информационные процессы, Том 17, № 2, 2017, стр. 101–113

МАТЕМАТИЧЕСКИЕ МОДЕЛИ, ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ

Две связанные задачи о сокращении избыточности при представлении данных с помощью выпуклых многогранников[1]

А.А.Бедринцев, В.В.Чепыжов

Институт проблем передачи информации, Российская академия наук, Москва, Россия

Поступила в редколлегию 08.06.2017

Аннотация—В работе рассмотрены две задачи о представлении данных с помощью выпуклых многогранников. В первой задаче требуется исключить из конечного множества X точек те из них, которые не являются вершинами многогранника выпуклой оболочки множества X. Разработан алгоритм, основанный на решении серии задач линейного программирования. Его вычислительная сложность асимптотически ниже сложности построения выпуклой оболочки, и ему требуется значительно меньший объем дополнительной памяти, чем при построении выпуклой оболочки. Вторая задача состоит в нахождении минимального подмножества неравенств в системе линейных неравенств, множество решений которого совпадает с множеством решений исходной системы. Показано, что эту задачу можно решать аналогично первой, и алгоритм решения обобщается на случай нелинейных неравенств. Предложены рандомизированные улучшения обоих алгоритмов.

КЛЮЧЕВЫЕ СЛОВА: линейное программирование, выпуклая оболочка, QuickHull, системы линейных неравенств, экстремальные эллипсоиды, эллипсоид Дикина, Hit-and-run, Billiard walk.

1. ВВЕДЕНИЕ

В работе рассмотрены две задачи о сокращении избыточности в представлении данных с помощью многогранников. Первая задача состоит в выделении из конечного множества точек минимального подмножества, имеющего ту же выпуклую оболочку, что и исходное множество. Эта задача эквивалентна построению вершин многогранника выпуклой оболочки. Вторая задача состоит в удалении из конечной системы линейных неравенств тех из них, которые не влияют на множество решений системы.

В процессе разработки сложных систем накапливаются данные с результатами расчетов характеристик конструируемых объектов при различных значениях параметров его геометрии и режимов функционирования. Эти данные используются в процессе последующих разработок. По результатам предыдущих точных, но длительных расчетов строится суррогатная модель, позволяющая в десятки тысяч раз быстрее производить необходимые вычисления характеристик нового объекта, допуская при этом погрешность в несколько процентов.

Суррогатные модели используются вместо точных функций при решении оптимизационных задач, в виде которых могут быть сформулированы инженерные задачи.

Для того, чтобы сформулировать такие оптимизационные задачи, необходимо указать множество допустимых значений для переменных. Естественно, эта область должна лежать недалеко от множества уже известных данных. Эта область должна иметь небольшой объем, содержать известные точки, быть выпуклой и, желательно, иметь простое описание. Этим требованиям удовлетворяют области в виде эллипсоидов [1, 2]. В качестве областей для оптимизации

[1] Работа выполнена при поддержке Российского научного фонда (проект № 14-50-00150).

можно рассматривать эллипсоиды минимального объема, содержащие заданные точки. При их построении можно ограничиться только теми точками, которые являются вершинами многогранника выпуклой оболочки всего множества данных. Действительно, в силу выпуклости эллипсоида, если он содержит вершины выпуклой оболочки набора точек, то он содержит и все остальные точки. Определение параметров эллипсоида сводится к задаче выпуклого программирования. Для каждой точки, которую должен содержать эллипсоид, строится соответствующее ограничение [3]. Поэтому актуальна задача выяснения, какие точки лежат в вершинах многогранника выпуклой оболочки множества, а какие без ущерба для задачи построения пространства допустимых значений можно отбросить.

Актуальность второй задачи состоит в следующем. На значения переменных вектора, представляющего реальный физический объект, накладываются ограничения, которые задают необходимые условия корректности объекта. Из предметной области известны границы изменения каждой отдельной величины и неравенства между различными параметрами. В работе рассматривается случай, когда данные ограничения могут быть сформулированы в виде системы линейных неравенств, решения которой образуют выпуклый многогранник. Заметим, что если даны нелинейные выпуклые ограничения, то они могут быть с необходимой точностью линеаризованы и представлены в виде системы линейных неравенств. Линейные неравенства входят в число функций-ограничений при решении задач оптимизации. Чем больше ограничений, тем большую вычислительную сложность имеют алгоритмы оптимизации. Возможна ситуация, когда система неравенств содержит лишние неравенства, удаление которых из системы не меняет многогранник множества ее решений. Если перед запуском процедуры оптимизации проверить, все ли неравенства в системе необходимы, и удалить лишние, то можно сократить время расчетов. Множество ограничений можно профильтровать однократно и пользоваться этим результатом при решении множества связанных оптимизационных задач.

Для описания и упрощенного рассмотрения ограниченного многогранника, заданного системой из M неравенств, часто применяется эллипсоид Дикина, который имеет следующие свойства:

1. Центр эллипсоида Дикина, который принято называть *аналитическим центром*, есть точка с максимальным произведением расстояний до гиперплоскостей всех граней;
2. этот эллипсоид принадлежит многограннику;
3. при растяжении эллипсоида Дикина в M раз относительно его центра получается эллипсоид, который содержит исходный многогранник [3].

Необходимо отметить, что эллипсоид Дикина зависит не от многогранника, а от системы неравенств. Можно доказать, что несколько раз добавляя в систему линейное неравенство, не меняющее ее множество решений, можно переместить аналитический центр многогранника в произвольную его внутреннюю точку (см. параграф 9). Такое изменение системы неравенств, не меняющее системы решений, влияет и на полуоси эллипсоида, т.е. на описание размеров многогранника. Для описания пространства допустимых значений с помощью эллипсоида Дикина нужно удалить избыточные неравенства, которые не меняют многогранник, но искажают эллипсоид.

Во многих пакетах уже реализованы функции построения выпуклой оболочки. Эти процедуры вместе с нахождением ее вершин определяют, какие вершины принадлежат одной грани многогранника. Результатом работы данных функций является список массивов, состоящих из номеров точек, принадлежащих каждой грани выпуклой оболочки. Задача перечисления граней по заданному набору вершин сама по себе вычислительно сложная и может требовать экспоненциально большое по размерности количество вычислений [4–6]. Для построения выпуклой оболочки известен алгоритм *Quick Hull*. Его сложность оценивается как

$O\left(N \log N + N^{\lfloor d/2 \rfloor}\right)$, где N – количество точек в d-мерном пространстве. Для решения поставленной задачи требуется дополнительное время на поиск уникальных номеров вершин среди полного описания многогранника. Также данный алгоритм требует большой объем дополнительной памяти, которая расходуется в том числе на перечисление всех граней многогранника.

В работе изложен метод решения задачи поиска минимального подмножества точек, образующих выпуклую оболочку исходного множества, с использованием серии задач линейного программирования. Данный метод не требует большой объем дополнительной памяти и время его работы слабо зависит от размерности. Предложен ряд рандомизированных процедур, ускоряющих решение задачи.

Удаление лишних неравенств из системы имеет схожий алгоритм решения с задачей об определении вершин выпуклой оболочки. Эта задача также решается с помощью серии задач линейного программирования, алгоритм решения строится по аналогии с первой задачей и допускает улучшения с помощью процедур *Hit-and-run* и *Billiard wolk*.

2. ПОСТАНОВКА ЗАДАЧИ НАХОЖДЕНИЯ ВЕРШИН ВЫПУКЛОЙ ОБОЛОЧКИ

Пусть имеется множество из N точек:

$$X = \{x_i \in \mathbb{R}^d\}.$$

Будем считать их попарно различными. Если встречаются дубликаты, то их можно удалить из X за $O\left(Nd \log N\right)$ операций: отсортируем X лексикографически (это потребует $O\left(N \log N\right)$ сравнений векторов длины d), после чего пройдем по массиву, сравнивая каждый вектор с последующим. Если есть дубликаты, то они находятся на соседних позициях. Поэтому для удаления всех повторяющихся точек достаточно одного прохода по массиву или $O\left(Nd\right)$ операций.

Рассмотрим все подмножества $Y \subseteq X$, такие, что их выпуклая оболочка совпадает с выпуклой оболочкой множества: $Conv\left(Y\right) = Conv\left(X\right)$. Выберем среди них минимальное по количеству точек в нем. Для краткости, назовем это подмножество *каркасом* множества X и обозначим $S\left(X\right)$.

Известно, что выпуклая оболочка конечного множества X представляет собой многогранник. Множество его вершин есть каркас $S\left(X\right)$. Следовательно, у конечного множества точек каркас существует и единственен.

Задача построения каркаса может быть сформулирована следующим образом:

$$\begin{aligned} &\min_{Y \subseteq X} |Y| \\ &s.t.\ Conv\left(Y\right) = Conv\left(X\right) \end{aligned}$$

3. АЛГОРИТМ РЕШЕНИЯ

В основе предлагаемого алгоритма лежит следующее наблюдение: точка x_1 принадлежит каркасу $S\left(X\right)$ тогда и только тогда, когда расстояние от нее до выпуклой оболочки остальных точек больше нуля:

$$\rho\left(x_1, Conv\left(X \backslash \{x_1\}\right)\right) > 0,$$

причем если найдется такое подмножество $Y \subset X$, $x_1 \notin Y$, что $\rho\left(x_1, Conv\left(Y\right)\right) = 0$, то $x_1 \notin S\left(X\right)$. Это следует из факта $Conv\left(Y\right) \subseteq Conv\left(X\right)$.

В конечномерном пространстве все нормы эквивалентны. В качестве нормы вектора $x = \left(x^1, \ldots, x^d\right)^T$ возьмем $\|x\|_\infty = \max\limits_j |x^j|$.

Пусть $Y = \{y_i\}_{i=1}^K$. Расстояние $\rho\left(x_1, Conv\left(Y\right)\right)$ может быть получено как значение z при решении следующей задачи линейного программирования:

$$\begin{aligned} &\min_{\alpha_1,\ldots,\alpha_K,z} z \\ &s.t. -z \le x_i^j - \sum_{i=1}^K \alpha_i y_i^j \le z,\ j = 1,\ldots,d; \\ &\alpha_i \ge 0,\ i = 1 \ldots K; \\ &\sum_{i=1}^K \alpha_i = 1. \end{aligned} \tag{1}$$

Действительно, точка $y = \sum\limits_{i=1}^K \alpha_i y_i$ при различных значениях $\alpha_i \ge 0$, $\sum\limits_{i=1}^K \alpha_i = 1$ (выпуклая линейная комбинация точек Y) пробегает выпуклую оболочку $Conv\left(Y\right)$. Тогда сформулированная задача есть задача поиска точки из $Conv\left(Y\right)$, ближайшей к x_1, и расстояния до нее в метрике $\|\cdot\|_\infty$, которое и есть минимальное значение величины z.

Предлагаются два алгоритма поиска каркаса: «Прогонка сверху» и «Прогонка снизу».

Прогонка сверху.

1. Положим $S\left(X\right) = X$.
2. Для каждой точки $x_i \in S\left(X\right)$:
   (a) вычислить $\rho_i = \rho\left(x_i, S\left(X\right) \setminus \{x_i\}\right)$;
   (b) если $\rho_i = 0$, то удалить x_i из $S\left(X\right)$.
3. $S\left(X\right)$ – искомый каркас.

Прогонка снизу.

1. Положим $S\left(X\right) = \{x_1\}$.
2. Для каждой точки $x_i \in X \setminus S\left(X\right)$:
   (a) вычислить $\rho_i = \rho\left(x_i, S\left(X\right)\right)$;
   (b) если $\rho_i \ne 0$, то:
      i. включить x_i в $S\left(X\right)$;
      ii. «валидация каркаса»: провести процедуру «Прогонки сверху» над $S\left(X\right)$.
3. $S\left(X\right)$ – искомый каркас.

Заметим, что алгоритм «Прогонка снизу» является открытым, т.е. позволяет обрабатывать точки, приходящие по одной, и для его работы не требуется знания всего множества данных до начала работы.

Для работы алгоритмов не требуется хранить информацию о всех гранях выпуклой оболочки. Объем дополнительной памяти ограничен числом точек в исходном множестве X и объемом, необходимым в процессе решения задачи линейного программирования.

4. АНАЛИЗ АЛГОРИТМОВ

Для анализа алгоритмов рассмотрим следующие наборы данных:

1. «Сфера»: генерируется N точек, лежащих на сфере единичного радиуса. Все точки этого множества, очевидно, принадлежат его каркасу.
2. «Симплекс»: выбирается $d+1$ точка каркаса $C_d = \{(0,\dots,0)\} \cup_{i=1}^{d} \{x_i : x_i^j = 0, j \neq i, x_i^i = 1\}$, и в множество данных добавляется $N-d-1$ точка внутри выпуклой оболочки $Conv\,(C_d)$. Данное множество имеет минимальный по размеру каркас, при этом выпуклая оболочка не лежит в подпространстве меньшей размерности.
3. «Шар»: генерируется N точек с многомерным нормальным распределением.

Для «Прогонки снизу» множество «Симплекс» даст лучший случай работы, если точки C_d будут расположены в начале множества. В этом случае будут решаться задачи не более чем с $d+1$ переменной, каркас не будет расти, увеличивая сложность последующих итераций.

Множество «Сфера» дает наихудший случай работы, т.к. каждая точка множества включается в каркас и требует проверки, не подлежит ли какая-то точка каркаса исключению из него.

Для «Прогонки сверху» множество «Симплекс» также дает наилучший случай, если точки C_d расположены в конце множества, так как на большинстве итераций происходит исключение точки из множества, следовательно, упрощение следующих итераций. При применении «Прогонки сверху» на множестве «Сфера» упрощения итераций не происходит, поэтому это множество также дает наихудший случай времени работы «Прогонки сверху».

Множество точек «Шар» моделирует средний случай работы алгоритмов.

В литературе широко освещаются задачи линейного программирования. Рассмотрим задачу линейного программирования в стандартном виде, в котором все ограничения имеют тип равенства, а переменные имеют неотрицательные значения. Наилучший известный авторам результат состоит в том, что решение задачи линейного программирования в стандартном виде с n переменными и m ограничениями имеет вычислительную сложность $O\left(\frac{n^{1.5}m^{1.5}}{\ln m}L\right)$, где L - длина битового представления задачи [7–9]. В стандартном виде задача (1) имеет вид:

$$\min_{\alpha_1,\dots,\alpha_K,z,\lambda_1,\dots,\lambda_d,\epsilon_1,\dots,\epsilon_d} z$$
$$s.t. x_1^j - \sum_{i=1}^{K} \alpha_i y_i^j + \lambda_j = z, j = 1,\dots,d;$$
$$-z + \epsilon_j = x_i^j - \sum_{i=1}^{K} \alpha_i y_i^j, j = 1,\dots,d;$$
$$\sum_{i=1}^{K} \alpha_i = 1,$$
$$z \geq 0,$$
$$\alpha_i \geq 0, i = 1 \dots K,$$
$$\lambda_i \geq 0, \epsilon_j \geq 0, j = 1 \dots d.$$

В этой задаче $m = 2d+1$ ограничений и $n = K + 2d + 1$ неотрицательных переменных. Неотрицательность переменной z следует из исходной системы ограничений.

Таким образом, сложность проверки на принадлежность точки каркасу при $K \gg d$ есть $O\left(\frac{K^{1.5}d^{1.5}}{\ln d}L\right)$. В «Прогонке сверху» всего необходимо выполнить N итераций, в которых $K \leq N$. Поэтому можно оценить вычислительную сложность алгоритма «Прогонка сверху» выражением $O\left(\frac{N^{2.5}d^{1.5}}{\ln d}L\right)$.

Вычислительная сложность «Прогонки снизу» сильно зависит от порядка, в котором на вход алгоритма подаются данные: если каркас часто обновляется, приходится вызывать процедуру «Прогонка сверху», подразумевающую решения задачи вида (1) для каждой точки текущего каркаса. Но если в процессе вычислений текущая точка включается в каркас на сравнительно малом числе итераций, то в процессе «Прогонки снизу» не решаются задачи линейного программирования с большим количеством переменных и ограничений, и выполняется малое количество процедур «валидации каркаса».

5. ЧИСЛЕННЫЕ ЭКСПЕРИМЕНТЫ

Проведено сравнение работы двух алгоритмов: предлагаемого «Прогонка сверху» и существующего *Quick hull*, реализованного в Matlab (функция *convhulln*). Задачи линейного программирования в процессе «Прогонки сверху» решались с помощью функции *linprog*.

Результат функции *convhulln* – матрица, где каждая строка содержит номера точек X, являющиеся вершинами одной грани выпуклой оболочки. Этот массив подвергается простой постобработке, в процессе которой определяется множество уникальных элементов, встречающихся в матрице.

Сравнение проводилось на компьютере с 3 Гб оперативной памяти. Эксперименты показали, что в многомерных пространствах ($d \geq 10$) построение выпуклой оболочки заканчивается с ошибкой нехватки памяти уже при $N = 200$ на множестве точек «Шар». При этом предлагаемый алгоритм «Прогонка сверху» успешно завершается при той же размерности входных данных и количестве $N = 400$ за время около минуты.

На практике алгоритм демонстрирует зависимость времени работы существенно лучше полученной теоретической оценки.

6. ПОСТАНОВКА ЗАДАЧИ УДАЛЕНИЯ ЛИШНИХ НЕРАВЕНСТВ

Пусть дан набор $P = \{p_i\}_{i=1}^N$ полупространств $p_i = \{x \in \mathbb{R}^d \mid a_i^T x \leq b^i\}$. Их пересечение образует выпуклый многогранник $M(P) = N(p_1, \ldots, p_N) = \cap_{i=1}^N p_i$, задаваемый системой линейных неравенств $M(P) = \{x \in \mathbb{R}^d \mid Ax \leq b\}$, где A – матрица размером $N \times d$, имеющая i-ю строку a_i^T, $b = \left(b^1, \ldots, b^N\right)^T$, и неравенства между векторами понимается покомпонентно. Будем предполагать, что множество $M(P)$ не пусто.

Рассмотрим такие подмножества $Q \subseteq P$, что $M(Q) = M(P)$, т.е. множества решений систем неравенств, задающих Q и P, совпадают. Неравенства, задающие элементы Q, естественно называть *актуальными*, а неравенства, соответствующие элементам $P \backslash Q$ - лишними.

Задача удаления всех лишних неравенств из системы может быть сформулирована в виде следующей задачи линейного программирования:

$$\begin{aligned} &\min_{Q \subseteq P} |Q| \\ &s.t.\ M(Q) = M(P) \end{aligned} \tag{2}$$

Для выявления лишних неравенств можно применить эллипсоид Дикина. Как отмечалось, при растяжении его полуосей в число раз, равное количеству неравенств в системе, он содержит многогранник. Следовательно, если гиперплоскость не пересекает растянутый эллипсоид Дикина, соответствующее неравенство является лишним [3].

Второй способ основан на решении системы линейных неравенств. У него достаточно сложное описание, которое можно найти в [10]. Задача с произвольной системой неравенств сначала сводится к однородной системе неравенств вида $c_i^T x \leq 0$, в которой все переменные принимают неотрицательные значения. Матрица коэффициентов транспонируется, каждому неравенству соответствует столбец в транспонированной матрице. Затем пары строк, отобранные по

некоторому правилу, заменяются на их линейную комбинацию с положительными коэффициентами. Если в матрице коэффициентов образуется столбец, в котором все коэффициенты неположительны, то соответствующее неравенство может быть исключено из системы, т.к. ему удовлетворяют любые неотрицательные числа, и оно зависимо от некоторого подмножества других неравенств.

Достоинством данных алгоритмов является возможность за одну итерацию удалить из системы сразу несколько лишних неравенств и небольшая алгоритмическая сложность. Второй алгоритм позволяет указать, следствием каких именно неравенств является данное лишнее неравенство. Но есть два существенных недостатка. Во-первых, данные алгоритмы не дают необходимого условия того, что ограничение является лишним. Если растянутый эллипсоид Дикина пересекается некоторой плоскостью, невозможно определить, пересекает ли она многогранник или проходит на небольшом расстоянии вне него. Во-вторых, данные методы не обобщаются на случай нелинейных ограничений.

Данная задача может быть решена с помощью алгоритмов, аналогичных приведенным в параграфах 3 и 4 алгоритмам «Прогонки снизу» и «Прогонке сверху».

Сформулируем условие, при котором неравенство $a_1^T x \leq b^1$ будет лишним. Найдем расстояние $\rho\left(\partial p_1, M\left(P \backslash \{p_1\}\right)\right)$ от границы данного полупространства до пересечения остальных полупространств как минимальное значение расстояния между двумя точками данных множеств:

$$\begin{aligned} &\min_{x,y} \|x-y\|_\infty \\ &s.t.\ x \in \partial p_1, \qquad y \in M\left(P \backslash \{p_1\}\right) = \cap_{i=2}^N p_i. \end{aligned} \tag{3}$$

Это расстояние может быть получено как значение z при решении следующей задачи линейного программирования:

$$\begin{aligned} &\min_{x,y \in \mathbb{R}^d,\, z \geq 0} z \\ &s.t. -z \leq x^j - y^j \leq z, j = 1, \ldots, d, \\ &a_1^T x = b^1, \\ &A_1 y \leq b_1 \end{aligned} \tag{4}$$

где A_1 – матрица, полученная из матрицы A удалением 1-ой строки, а вектор b_1 получен удалением 1-ого элемента из вектора b.

Если полученное расстояние положительно, то соответствующая гиперплоскость не пересекает многогранник, образованный остальными полупространствами. А значит, лежит на некотором расстоянии от него, и соответствующее неравенство является лишним.

Вышеизложенный метод требует уточнения для частного случая, когда лишнее ограничение проходит через вершины многогранника (см. рис. 1). Одно из ограничений является лишним, хотя и проходит через точку пересечения двух прямых, соответствующих существенным неравенствам.

Такие неравенства можно удалять с помощью утверждения, сформулированного в [11]. Обозначим через $V(x) = \{a_i : a_i^T x = b^i\}$ множество векторов нормалей к тем гиперплоскостям, на которых лежит точка x (соответствующие неравенства являются *активными* в точке x). Справедлива

Теорема 1. *Если некоторый вектор $a_k \in V(x)$ раскладывается по остальным векторам из $V(x)$ с неотрицательными коэффициентами, то неравенство $a_k^T \leq b^k$ является лишним.*

На базе критерия (4) формулируется алгоритм «Прогонки сверху» для решения задачи (2).

Алгоритм прогонки сверху.

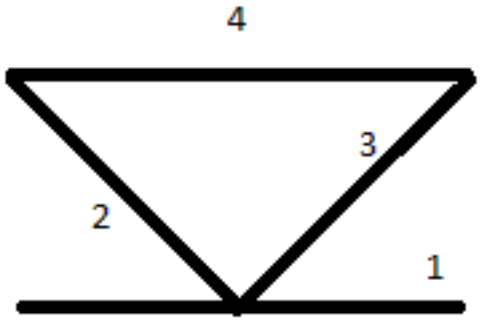

Рис. 1. Ограничение №1 является лишним, но расстояние до многогранника, образованного отрезками 2-4 равно нулю

1. Для каждого полупространства $p_i \in P$:
   (a) Проверить, является ли оно лишним, с помощью задачи типа (4).
   (b) Проверить выполнение условий теоремы 1 для других неравенств, активных в точке минимума, найденной на предыдущем шаге.
   (c) Если ДА, то удалить его из системы: $P = P \setminus \{p_i\}$.
2. $Q = P$ – искомое множество актуальных неравенств.

Аналогично можно сформулировать алгоритм «Прогонки снизу» для нахождения минимального подмножества существенных неравенств в системе.

Приводя задачу (4) к стандартному виду и предполагая, что $d << N$, получим задачу линейного программирования с $O(N)$ переменными и $O(N)$ ограничениями. Сложность проверки одного неравенства может быть оценена как $O\left(\frac{N^3}{\ln N}L\right)$, где L - длина битового представления задачи (4) в стандартном виде. «Прогонка сверху» имеет сложность $O\left(\frac{N^4}{\ln N}L\right)$. Вычислительная сложность алгоритма «Прогонки сверху» существенно зависит от порядка следования неравенств в системе.

Возможность обобщения предложенного метода исключения лишних неравенств из системы на случай нелинейных ограничений следует из того, что для решения задачи, сформулированной в виде (3), не требуется линейности границ полупространств p_i.

7. УЛУЧШЕННЫЙ И РАНДОМИЗИРОВАННЫЙ АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ О ВЫПУКЛОЙ ОБОЛОЧКЕ

У предложенных алгоритмов есть потенциал для улучшения. Задача (1) может решаться с учетом ее особенностей. Во-первых, можно указать точку, удовлетворяющую всем ограничениям задачи (1). При достаточно большом значении z_0 это точка $\left(\alpha_1 = \frac{1}{K}, \ldots, \alpha_K = \frac{1}{K}, z_0\right)$. Во-вторых, само расстояние находить не требуется, а нужно лишь проверить, равно оно нулю или нет. В процессе работы алгоритма на некоторой итерации может быть получена оценка погрешности текущего значения по сравнению с оптимальным. Если ноль с большой вероятностью не попадает в доверительный интервал, то решение задачи линейного программирования может быть досрочно прервано, а точка включена в каркас.

При «Прогонке снизу» можно уменьшить количество выполнения шагов 2.b.ii следующим образом. Сложность каждой итерации связана с мощностью текущего каркаса. Пусть текущая точка подлежит включению в каркас. Процедура валидации проверяет, действительно

ли необходимо усложнять последующие итерации алгоритма, или каркас можно сократить, удалив одну или несколько точек каркаса с предыдущей итерации, оказавшуюся внутри выпуклой оболочки нового каркаса. Однако, процедура валидации достаточно дорогая, и исключение точек происходит не обязательно (особенно в многомерном пространстве). Можно исследовать вопрос об условии, при котором можно пропускать некоторое количество раз процедуру валидации каркаса, и проводить ее после добавления нескольких точек в каркас. Оно будет зависеть от текущей мощности каркаса, количества необработанных точек, и текущего количества пропущенных процедур валидации.

Если известно все множество данных X, то можно предложить еще одну модификацию «Прогонки снизу». В качестве начального каркаса можно взять набор из точек, у которых одна из координат имеет минимальное или максимальное значение среди всех точек выборки. Обобщая, можно выбрать несколько векторов $n_1, \ldots, n_k$ и включить в начальный каркас точки, на которых достигается $\min\limits_j n_i^T x_j$ и $\max\limits_j n_i^T x_j$. Заметим, что найти эти точки можно за $O(Ndk)$ операций. Данные точки принадлежат каркасу и будут покрывать большую часть выпуклой оболочки, следовательно, новые точки в каркас будут добавляться реже, и «Прогонка сверху» над текущим каркасом будет выполняться менее часто. Если сразу несколько точек из X дают одинаковые экстремальные значения одних и тех же линейных форм, то можно провести «Прогонку сверху» над начальным каркасом.

Подобная процедура возможна и в задаче удаления лишних ограничений. Подробнее она рассмотрена в следующем параграфе.

8. МОДИФИКАЦИЯ МЕТОДА РЕШЕНИЯ ЗАДАЧИ О НЕРАВЕНСТВАХ НА БАЗЕ ПРОЦЕДУРЫ HIT-AND-RUN

Предлагается рандомизированная модификация алгоритма, позволяющая его ускорить, а в случае, когда большинство неравенств системы являются существенными ($|P \setminus Q| = o(N)$), она позволяет уменьшить асимптотику времени работы алгоритма.

Известна процедура Hit-and-run [12] генерации случайных точек внутри многогранника (см. рис. 2). Пусть дан выпуклый многогранник $M(P)$ и некоторая его внутренняя точка $x_0 \in int\ M(P)$.

Алгоритм Hit-and-run.

1. Для $k = 1, 2, \ldots$:
   (a) Сгенерировать случайный вектор r_k.
   (b) Для прямой $x = x_{k-1} + r_k t$ найти все точки пересечения с гиперплоскостями из P. Необходимо решить линейные уравнения $a_i^T(x_{k-1} + r_k t_i^k) = b^i$.
   (c) Найти гиперплоскости с наименьшими по модулю неположительными и неотрицательными t_i^k: $t_{\alpha_k}^k = -\min\limits_{i:t_i^k<0} |t_i^k|$, $t_{\beta_k}^k = \min\limits_{i:t_i^k>0} |t_i^k|$ (если отрицательного t_i^k не существует, то $t_{\alpha_k}^k$ полагается равным минус бесконечности, если положительного t_i^k не существует, то $t_{\beta_k}^k$ полагается равным плюс бесконечности).
   (d) Выбрать случайное число $p_k \in [t_{\alpha_k}^k, t_{\beta_k}^k]$, положить $x_k = x_{k-1} + r_k p_k$.
2. $x_1, x_2, \ldots$ – случайные точки в многограннике.

«Посещенные» гиперплоскости, соответствующие индексам α_k и β_k, являются существенными и могут не проверяться с помощью процедуры (4). Однако про непосещенные гиперплоскости ничего сказать нельзя. Они могут соответствовать действительно лишним неравенствам, и до них с помощью процедуры Hit-and-run невозможно дойти, а могут быть существенными гипергранями малого объема, вероятность попасть в которые мала.

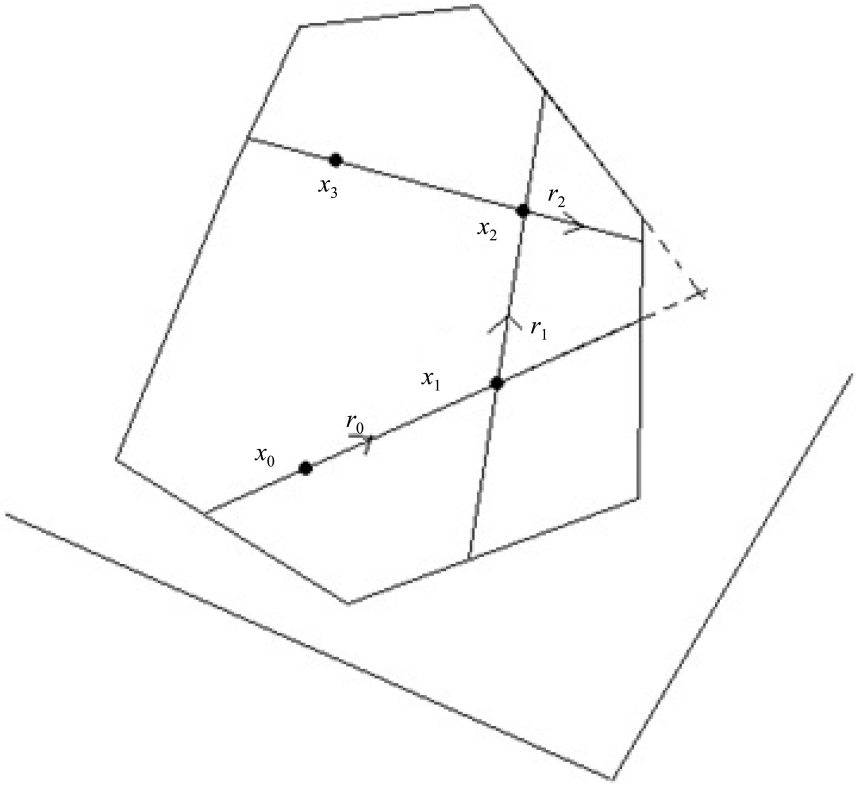

Рис. 2. Алгоритм *Hit and run* генерации случайных точек в выпуклом многограннике

Метод Hit-and-run вычислительно эффективен. Сложность генерации одной точки – решения N линейных уравнений с d подобными слагаемыми и нахождения минимума и максимума среди N чисел (пренебрежем случаями, когда некоторые уравнения не имеют корней) – $O(Nd)$. Таким образом, без изменения асимптотики вычислительной сложности алгоритма можно сделать до $O(N^3)$ итераций процедуры Hit-and-run.

При генерации одной точки посещаются две гиперплоскости. Как показывает численный эксперимент, если предположить равную вероятность посещения существенных граней многогранника (например, правильный многоугольник на плоскости), то для посещения всех существенных граней достаточно сделать $O(|Q|)$ итераций. А именно, за число итераций, превышающее количество существенных граней в 5-6 раз, 99% существенных гиперплоскостей будут посещены (см. рис. 3).

Также известен более продвинутый метод блуждания по многограннику Billiard walk [13]. Доказана его сходимость к равномерному распределению. Его отличие от метода Hit-and-run в том, что на каждом шаге генерируется длина траектории. Если точка, двигаясь в случайном направлении, достигает границы многогранника, она отражается по правилу «угол падения равен углу отражения» и продолжает движение либо до следующей грани, либо до конца траектории. Конец траектории берется за новую сгенерированную точку. Выбирается случайное направление, длина траектории и процесс повторяется. В этом случае все посещенные гиперплоскости, от которых происходило отражение, помечаются как существенные. Для посещения одной гиперплоскости методом Billiard walk требуется примерно столько же операций, сколько в методе Hit-and-run.

Заметим, что из представленных оценок следует, что процедура получения приближенного решения задачи (2) удаления лишних неравенств на базе метода генерации случайных точек

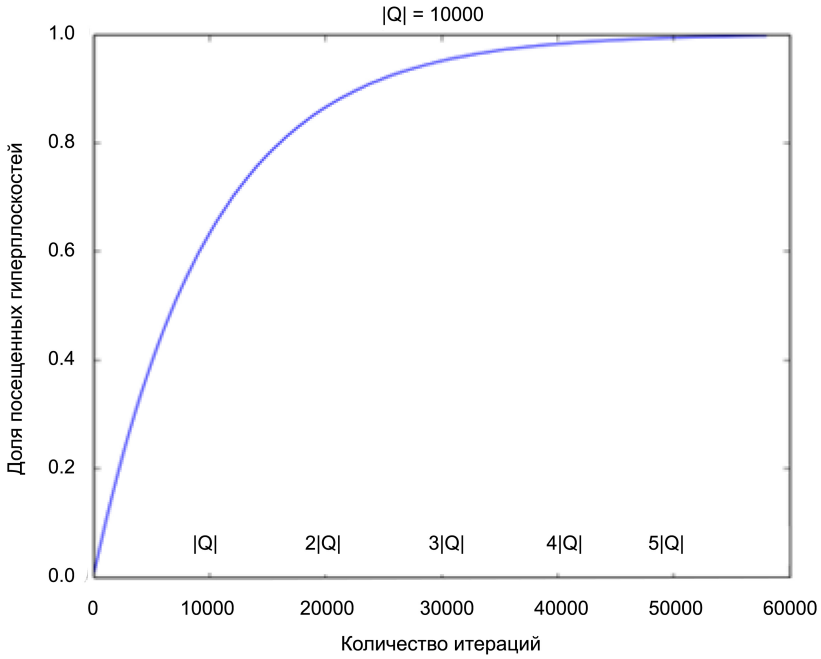

Рис. 3. Количество посещенных гиперплоскостей в процессе работы алгоритма *Hit and run* в предположении равновероятности попадания в каждую из Q гиперплоскостей

внутри многогранника имеет сложность $O(|Q|Nd) \subseteq O(N^2 d)$ операций. Для получения точного решения необходимо на втором этапе непосещенные гиперплоскости проверить с помощью задач типа (4). В случае, когда имеется значительное количество существенных неравенств, большинство из них может быть посещено, после чего останется проверить $o(N)$ неравенств. В этом случае время работы алгоритма асимптотически меньше оценки для базовой прогонки сверху.

9. ПРИЛОЖЕНИЕ

Докажем, что, добавляя в совместную систему неравенств лишние неравенства, можно аналитический центр системы неравенств переместить в произвольную внутреннюю точку многогранника решений, если он имеет ненулевой объем. Это утверждение сформулировано в [3] без доказательства.

Теорема 2. *Дан непустой, ограниченный и невырожденный многогранник* $M(P) = \{x \in \mathbb{R}^d \| Ax \leq b\}$, $int(M(P)) \neq \emptyset$, $\exists C : \forall x \in M : |x| \leq C$, *его произвольная внутренняя точка* $x_0 \in int(M(P)) \Leftrightarrow Ax_0 < b$. *Доказать, что* $\exists c \in \mathbb{R}^d, \gamma \in \mathbb{R}, q \in \mathbb{N} \cup \{0\} : M(P) = M\left(P \cup \{x \in \mathbb{R}^d \| c^T x \leq \gamma\}\right)$ *и* x_0 *- аналитический центр системы неравенств* $\{Ax \leq b, c^T x \leq \gamma, \ldots, c^T x \leq \gamma\}$, *где последнее неравенство повторено* q *раз.*

Доказательство. Рассмотрим логарифмическую барьерную функцию системы с добавленными неравенствами:

$$\phi(x_0) = -\sum_{i=1}^{N} \ln\left(b^i - a_i^T x_0\right) - q \ln\left(\gamma - c^T x\right)$$

Аналитический центр – решение векторного уравнения

$$\nabla\phi\left(x_0\right)=\sum_{i=1}^{N}\frac{1}{b^i-a_i^Tx_0}a_i+q\frac{1}{\gamma-c^Tx}=0,$$

которое перепишем в виде:

$$\sum_{i=1}^{N}\frac{1}{b^i-a_i^Tx_0}a_i=q\frac{1}{c^Tx-\gamma}. \tag{5}$$

Требуется показать, что оно имеет решение с дополнительными ограничениями $q\in\mathbb{N}\cup\{0\}$ и $M\left(P\right)=M\left(P\cup\{x\in\mathbb{R}^d\|c^Tx\leq\gamma\}\right)$.

Обозначим левую часть уравнения (5) -– некоторый постоянный вектор -– через z.

Точка $x_0=x_{ac}$ является аналитическим центром системы неравенств P, тогда и только тогда, когда $z=0$.

Если $z=0$, то $c=0,q=0,\gamma=0$ решает исходную задачу, то есть аналитический центр уже находится в требуемой точке и никакое неравенство добавлять в систему не нужно.

Пусть $z\neq0$. Тогда $q\neq0$ (добавлять неравенство необходимо) и из (5) $c\neq0$.

Перепишем (5) в виде системы из d линейных уравнений относительно d компонент вектора c.

$$z\left(c^Tx_0-\gamma\right)=qc.$$

Система может быть переписана в виде:

$$z^j\left(\sum_{i=1}^{d}c^ix_0^i-\gamma\right)-gc^j,j=1\ldots d$$

$$z^j\sum_{i=1}^{d}c^ix_0^i-qc^j=z^j\gamma,j=1\ldots d$$

$$\sum_{i=1}^{d}\left(q\delta_{ij}-z^jx_0^i\right)=-z^j\gamma,j=1,\ldots,d.$$

При достаточно большом q матрица системы будет обладать строгим диагональным преобладанием (т.е. диагональный элемент будет по модулю больше суммы модулей остальных элементов той же строки матрицы). Следовательно, матрица системы будет невырожденной [14], решение у системы существует и единственно при любом γ.

Перепишем систему в виде: $\left(qI-B\right)c=-z\gamma$, где $B_{ji}=z^jx_0^i$. В силу доказанного, можно считать $q\neq0$ и $c=-\frac{1}{q}\left(I-\frac{1}{q}B\right)^{-1}z\gamma$.

Расстояние от начала координат до гиперплоскости $c^Tx=\gamma$ равно $\rho=\frac{|\gamma|}{|c|}$.

Оценим норму $|c|$:

$$|c|=\frac{|\gamma|}{q}\left|\left(I-\frac{1}{q}B\right)^{-1}z\right|\sim\frac{|\gamma|}{q}\left|z\right|,q\to\infty.$$

Следовательно, $\rho=\frac{q}{|z|}\to\infty$, и гиперплоскость может быть удалена сколь угодно далеко от начала координат. Ограниченный многогранник $M\left(P\right)$ лежит по одну сторону от нее.

Покажем, что можно выбрать параметры так, что $c^Tx_0\leq\gamma$, что докажет, что добавляемое ограничение не меняет решение исходной системы неравенств, если $x_0\in intM\left(P\right)$.

$$c^Tx_0=-\gamma z^T\left(I-\frac{1}{q}B^T\right)^{-1}\frac{x_0}{q}$$

При $q \to \infty$ $c^T x_0 \sim \frac{-\gamma z^T x_0}{q}$. Выберем $\gamma = 1$. Тогда независимо от значения скалярного произведения $z^T x_0$, которое фиксировано при выборе x_0, при достаточно большом q неравенство будет выполняться $c^T x_0 \le \gamma$. И, напомним, при достаточно большом q вектор c существует, причем с увеличением q растет расстояние от начала координат до гиперплоскости, соответствующей добавляемому неравенству. □

СПИСОК ЛИТЕРАТУРЫ

1. Бедринцев А.А., Чепыжов В.В. Двухкритериальная задача построения оптимальных эллипсоидов для представления данных. *Информационные технологии и системы*, Нижний Новгород, 2014.
2. Бедринцев А.А. Представление данных с помощью минимальных эллипсоидов. *Труды 56-й научной конференции МФТИ «Актуальные проблемы фундаментальных и прикладных наук в современном информационном обществе»*. 2013, т. 2, С. 92–94.
3. Boyd S., Vandenberghe L. *Convex Optimization*. Cambridge: UniversityPress, 2004.
4. Bradford Barber C. [et al.]. The Quickhull Algorithm for Convex Hull . *ACM Transactions on Mathematical Software*. Vol. 22, No. 4. 1996, pp. 469–483.
5. Bernard Chazelle. An Optimal Convex Hull Algorithm in Any Fixed Dimension. *Discrete & Computational Geometry*, Vol. 10, 1993, pp. 377–409.
6. Препарата Ф., Шеймос М. *Вычислительная геометрия: Введение*. М.: Мир, 1989.
7. Нестеров Ю.Е. *Методы выпуклой оптимизации*. М.: «МЦМНО», 2010.
8. Хачиян Л.Г. *Сложность задач линейного программирования*. М.: Знание. 1987 (Новое в жизни, науке и технике. Сер. «Математика, кибернетика»; №10).
9. Anstreicher K. Linear Programming in $O\left(\frac{n^3}{\ln n} L\right)$ operations. *SIAM J. on Optimization*. Vol. 9, No 4, 1999, pp. 803–812.
10. Черников С.Н. *Линейные неравенства*. М.: Наука, 1968.
11. Caron R.J., McDonald J.F., Ponic C.M. A degenerate extreme point strategy for the classification of linear constraints as redundant or necessary. *Journal of Optimization Theory and Applications*. Vol. 62, No. 2, 1989, pp. 225–237.
12. Smith R.L. Efficient Monte-Carlo Procedures for Generating Points Uniformly Distributed over Bounded Regions. *Operations Research*. Vol. 32. 1984. pp. 1296–1308.
13. Gryazina E., Polyak B. Random sampling: Billiard Walk algorithm // European Journal of Operational Research. Vol. 238. 2014. pp. 497–504.
14. Roger A. Horn & Charles R. Johnson. *Matrix Analysis*. Cambridge: Cambridge University Press, 1985.

Two related problems regarding redundancy reduction for data represenation using convex polyhedra

Bedrintsev A., Chepyzhov V.

Two problems on data representation using convex polyhedra are discussed in the article. The first problem is to remove from a finite set of points those which are not vertices for their convex hull. Algorithm based on solving of series of linear programming problems is developed. Its computational complexity and required memory is asymptotically lower than complexity of convex hull searching. The second problem is to find minimal subset of inequalities in the system of linear inequalities that gives the same solution as the entire system. It is shown that this problem is solved similarly to the first one and the algorithm is generalized to the system of nonlinear inequalities. Randomized improvements of both algorithms are suggested.

KEYWORDS: linear programming, convex hull, QuickHull, systems of linear inequalities, extremal ellipsoids, Dikin ellipsoid, Hit-and-run, Billiard walk.