Тестирование алгоритма, максимизирующего объем вписанного многогранника, в среде высокопроизводительных вычислений.

Кокорев Денис Сергеевич
Аспирант ИППИ РАН
korvin-d@yandex.ru

Аннотация

В статье рассматривается проблема нахождения многогранников заданной формы внутри других многогранников. Предлагаются несколько альтернативных методов поиска вписанных многогранников, основанных на сведении данной задачи к задаче нелинейного программирования и решения ее с помощью готовых программных вычислительных ресурсов. Также предлагаются возможные аппроксимации для ускорения работы алгоритма. Описывается система распределенного тестирования, использующаяся для развития алгоритма.

1. Введение

Одной из классических математических проблем является третья часть 18-ой проблемы Гильберта. В своей общей постановке она посвящается упаковкам одних тел другими. В случае упаковки правильными объектами, например сферами, существуют доказанные результаты, например: сильная проблема тринадцати сфер[1] и гипотеза Кеплера[2]. В случае упаковки многогранников в другие многогранники, задача оказывается очень сложной, и пока не существует теоретических подходов, позволяющих подступиться к этой задаче в общем виде.

Частным случаем этой проблемы является задача нахождения в многограннике произвольной формы, возможно не выпуклом, объекта заданной формы с наибольшим объемом. Данная задача имеет широкую область применения: компьютерное моделирование трехмерных объектов; программные тренажеры; компьютерные игры; приложения, решающие задачи упаковки и раскроя; программы, отвечающие за передвижение роботов; ювелирная промышленность; обработка дорогостоящих материалов.

Первоначальным этапом алгоритма, решающего эту задачу, является нахождение стартовой точки - начального приблизительного расположения объекта. В качестве стартовой точки могут быть взяты данные работы других неточных алгоритмов. Одним из примеров таких алгоритмов является гомотетичное раздувание под разными углами и с центром масс в разных точках искомого объекта до пересечения с внешним многогранником и нахождение, таким образом, стартовой точки с максимальным объемом. Далее необходимо шевеля вершины на незначительные расстояния найти положение с максимальным объемом и удовлетворяющие некоторым требованиям на форму.

2. Используемые термины

Определение 1. *Вписыванием* одного многогранника в другой будем называть нахождение многогранника, комбинаторно эквивалентного первому, максимального объема и содержащегося во втором.

Определение 2. Будем говорить, что два многогранника обладают одинаковой *комбинаторной структурой* тогда и только тогда, когда изоморфны их граничные комплексы[3]. Такие многогранники называются *комбинаторно эквивалентными*.

Определение 2.1. *Комплексом* называется конечная совокупность K многогранников в E_d, удовлетворяющая условиям: 1) наряду с каждым многогранником M из семейства K в K входит также и любая грань многогранника M; 2) пересечение любых двух многогранников из K является гранью каждого из них.

Определение 2.2. Пусть M — d-многогранник (размерности d) в E_d , и пусть целое число k

удовлетворяет условию $0 < k < d$. Множество всех граней многогранника M размерности, не превышающей k, является комплексом, который называется *k-скелетом многогранника M.*

Определение 2.3. (d - 1)-скелет многогранника M будем обозначать символом $F(M)$ и называть *граничным комплексом многогранника*.

Определение 2.4. Два комплекса K и K' называются *изоморфными комплексами*, если между ними существует взаимно однозначное отображение φ, сохраняющее операцию включения: $F_1 \subset F_2 \Leftrightarrow \varphi(F_1) \subset \varphi(F_2)$

Определение 3. *Задачей нелинейного программирования (НП-задача)* называется оптимизационная задача следующего вида:

$f(x) \rightarrow min$

при ограничениях:

$gL \leq g(x) \leq gU$

$xL \leq x \leq xU$

где $x \in \mathbb{R}^n$ – оптимизационные переменные с верхними и нижними ограничениями:

$xL \in (\mathbb{R} \cup \{-\infty\})^n$, $xU \in (\mathbb{R} \cup \{+\infty\})^n$

$f: \mathbb{R}^n \rightarrow \mathbb{R}$ - целевая функция

$g: \mathbb{R}^n \rightarrow \mathbb{R}$ - общие нелинейные ограничения с верхними и нижними границами:

$gL \in (\mathbb{R} \cup \{-\infty\})^m$, $gU \in (\mathbb{R} \cup \{+\infty\})^m$

$f(x)$ и $g(x)$ могут быть линейными или нелинейными, выпуклыми или невыпуклыми.

Определение 4. *AMPL* [4] (*A Modeling Language for Mathematical Programming*) — язык программирования высокого уровня для описания и решения сложных задач оптимизации и теории расписаний. Главным преимуществом AMPL является подобие его синтаксиса математической записи задач оптимизации, что позволяет дать очень краткое и легко читаемое определение задач математического программирования. Для решения задач, написанных на AMPL, используются вычислительные солверы.

Определение 5. *Нелинейный солвер* – программа для решения задач нелинейного программирования, использующая один из алгоритмов таких как: метод градиентного спуска, метод внутренней точки или квазиньютоновские методы.

Определение 6. *Выпуклой оболочкой* множества X называется наименьшее выпуклое множество, содержащее X.

Определение 7. *Смешанным произведением* $(\vec{a}, \vec{b}, \vec{c})$ векторов $\vec{a}, \vec{b}, \vec{c}$ называется скалярное произведение вектора $\vec{a}$ на векторное произведение векторов $\vec{b}$ и $\vec{c}$.

$$(\vec{a}, \vec{b}, \vec{c}) = \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix}$$

Ориентированный объём симплекса в трехмерном евклидовом пространстве можно определить по формуле:

$V = \frac{1}{6} det\,(v_1 - v_0, v_2 - v_0, v_3 - v_0)$, где v_i – вектора координат вершин симплекса.

3. Постановка задачи

Даны два многогранника – внешний невыпуклый и внутренний выпуклый. Внутренний многогранник является начальным приблизительным решением задачи и содержится внутри внешнего. Требуется максимально увеличить, шевеля вершины, внутренний многогранник так, чтобы он остался вписанным и сохранил свою комбинаторную структуру. Помимо комбинаторной структуры могут быть добавлены соотношения каких-либо размеров фиксирующих форму многогранника необходимые для прикладного использования алгоритма.

4. Первый метод

Поставленную геометрическую задачу необходимо записать в терминах задачи нелинейного программирования.

В качестве целевой функции $f(x)$ в нашем случае берется объем внутреннего многогранника. Он вычисляется через координаты вершин с помощью разбиения граней на треугольники и представление объема в виде суммы объемов симплексов. Если в прикладном применении метода ценность искомого объекта зависит не только от объема, то эти характеристики также можно внести в целевую функцию.

Переменными в нашей задаче будут параметры, отвечающие за расположение вершин в пространстве, $x_{ji} \in (-\infty, +\infty)$, где j – номер точки, i – ось координат. Если есть уверенность, что начальное расположение объекта близко к искомому, то на координаты вершин можно наложить ограничения такие, чтобы вершины не удалялись от начального

состояния дальше, чем на некоторую величину *dx*.

Перейдем к описанию общих ограничений *g(x)*. Есть несколько основных групп ограничений для данной задачи, без которых нельзя обойтись. Первой группой таких ограничений являются факты, что вершины должны остаться на своих гранях. Если в грани было три вершины, то все заведомо в порядке. Если же больше, то требуется наложить ограничения. Условие того, что четыре точки лежат в одной плоскости равносильно тому, что объем натянутого на них симплекса равен нулю, то есть в нерасписном виде условие выглядит так:

$det\ (x_1 - x_0, x_2 - x_0, x_3 - x_0) = 0$

где x_0 , x_1, x_2, x_3 – точки, лежащие на одной грани. Для лучшей точности это условие необходимо записать для всех четверок вершин во всех гранях, но это сильно замедлит время вычислений солвера, поэтому можно оптимизировать работу исходя из того, что некоторые наборы точек лежат именно в одной грани. Поэтому можно в каждой грани брать фиксированные три точки и добавлять к ним все остальные по очереди.

Чтобы не возникало пересечения граней по ребрам, которых не должно существовать, необходимо записать условия выпуклости внутреннего многогранника. При данном подходе единственным способом задания ограничений на выпуклость многогранника является выписывание условий локальной выпуклости для каждой из вершин. Для этого необходимо для каждой вершины брать тройки смежных с ней по ребрам вершин и смотреть объем натянутого на них симплекса. Если вектора к этой тройке вершин образуют правую декартову систему координат и начальная вершина является выпуклой, то объем симплекса будет больше нуля, если же точка не выпукла, то объем отрицательный. Исходя из этого, получаются следующие условия:

$det\ (x_1 - x_0, x_2 - x_0, x_3 - x_0) \geq 0$

только теперь уже x_1, x_2, x_3 – точки смежные по ребрам с x_0. Если было введено ограничение на сдвиг вершин *dx*, то для оптимизации по времени эти условия можно записывать только для групп близко лежащих вершин, расстояние между которыми соизмеримо с *dx*, потому что прочие вершины не смогут достаточно сильно поменять местонахождение в пространстве, чтобы появились невыпуклые дефекты.

Последняя группа обязательных условий связана с тем, что внутренний многогранник должен остаться вписанным. Эти условия записываются в следующем виде:

$A_r x_{j1} + B_r x_{j2} + C_r x_{j3} - D_r \leq 0$

где *r* - номер грани внешнего многогранника, а *j* – номер вершины внутреннего многогранника. Стоит заметить, что A_r ,B_r ,C_r ,D_r являются константами, а не переменными; а значит, эта группа ограничений является линейными и не значительно влияет на время работы солвера.

Помимо обязательных ограничений, в зависимости от конкретной задачи, могут быть добавлены и другие ограничения, связанные с какими-то размерами многогранника. Но не стоит забывать, что чем выше степень многочлена, описывающего ограничения, тем сильнее оно замедлит работу солвера.

Этот метод является неэффективным, потому что возникает очень много сложных ограничений третьего порядка, что значительно замедляет работу солвера. Также возникают сложности с записью дополнительных ограничений, связанных с нормалями граней. Преимуществом данного метода является минимальное возможное количество используемых переменных. Это делает матрицы производных меньшего размера, но при этом они являются менее разреженными.

Описанный выше метод был признан не эффективным на стадии, когда внешний многогранник брался только выпуклым. Для не выпуклого случая необходимые ограничения будут описаны в главе 5.1.

5. Второй метод

Главной идеей второго является введение дополнительного набора переменных, отвечающих за параметры граней внутреннего многогранника. Есть несколько преимуществ данного подхода. Во-первых, порядок большей части уравнений в задаче сокращается с третьего на второй, что значительно уменьшает время работы. Во-вторых, уравнения, записанные с использованием двойного набора переменных, являются более понятными с геометрической точки зрения, что значительно сокращает время отладки алгоритма. В-третьих, существует масса дополнительных ограничений, например связанных с углами многогранника, которые просто невозможно записать без введения дополнительного набора переменных.

Итак, переменные во втором методе будут делиться на 2 типа. Первый тип – параметры граней внутреннего многогранника y_{ap} , y_{bp} ,y_{cp} ,

y_{dp} ,где p – номер грани внутреннего многогранника. В качестве ограничений (xU и xL) на переменные первого типа берутся начальные значения параметров плюс/минус некоторая величина соответственно. Размер этой величины зависит от того, насколько разрешается изменять углы нормалей граней заданной комбинаторной структуры. Параметры граней внешнего многогранника не меняются, поэтому рассматриваются как константы. Второй тип переменных – это параметры, отвечающие за расположение вершин внутреннего многогранника в пространстве, $x_{ji} \in (-\infty, +\infty)$, где j – номер точки, i – ось координат, аналогично первому методу.

Ограничения на комбинаторную структуру становятся вида:

$y_{ap}x_{j1} + y_{bp}x_{j2} + y_{cp}x_{j3} - y_{dp} = 0$

где p – номер грани внутреннего многогранника, j – номер вершины, индексы *1,2,3* соответствуют трем осям координат. Рассмотренная группа ограничений записывается для всех пар вершина-грань, для которых верно, что вершина лежит на грани. Перечень всех таких пар получается из начального представления граней многогранника.

Ограничения на выпуклость необходимо записать для всех пар вершина-грань внутреннего многогранника, для которых вершина не принадлежит грани:

$y_{ap}x_{j1} + y_{bp}x_{j2} + y_{cp}x_{j3} - y_{dp} \leq 0$

5.1. Невыпуклый случай

Для случая, когда внешний многогранник является невыпуклым, необходимы дополнительные ограничения, отделяющие внутренний многогранник от не выпуклых частей внешнего многогранника. Сначала строится выпуклая оболочка внешнего многогранника. В описанных в главе 5 ограничениях используются плоскости этой выпуклой оболочки. Далее необходимо вычислить какие грани базового внешнего многогранника являются невыпуклыми. Это делается, используя правило, что грань является невыпуклой, если в многограннике есть две вершины, лежащие по разные стороны от этой грани. Далее для каждой такой невыпуклой грани необходимо создать отдельную разделяющую плоскостью. Разделяющая плоскость задается свободными переменными y_a, y_b, y_c, y_d. На эту разделяющую плоскость накладываются ограничения, что все вершины невыпуклой плоскости, привязанной к ней, лежат по одну сторону от нее. А все вершины вписываемого многогранника лежат по другую сторону от нее. Такие уравнения задаются для каждой невыпуклой грани внешнего многогранника. Если для вершин используется максимальное отклонение от начального положения *dx,* то можно отделять разделяющей плоскостью не все вершины вписываемого многогранники, а только вершины тех граней, которые могут пересечься с невыпуклой гранью внешнего многогранника. Зная значение *dx* не сложно вычислить набор таких граней для каждой невыпуклой грани внешнего многогранника. Для упрощения работы солвера можно дополнительно задать ограничение, что нормаль разделяющей плоскости близка к 1.0.

6. Вычисления на солвере

Для вычислений сформулированной НП-задачи мной был использован солвер IPOPT[5](Internal Point OPTimization). Это солвер предназначен для поиска локального оптимума в задаче нелинейного программирования методом внутренней точки. Для вычисления солверу необходимо предоставить callback-функции, вычисляющие следующие величины:

- значения целевой функции $f(x)$;
- градиента целевой функции $\nabla f(x)$;
- вектора значений вектор-функции ограничений $g(x)$;
- якобиана вектор-функции ограничений $\nabla g(x)^{\mathrm{T}}$
- гессиана расширенной функции Лагранжа $\sigma_f \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x)$

Если для формулировки задачи использовать язык программирования AMPL, то он сам предоставляет эти функции при получении $f(x)$ и $g(x)$. Но AMPL является платным и дорогостоящим для коммерческого использования, поэтому был разработан и реализован интерфейс для работы в солвером Ipopt напрямую, без использования AMPL. Были написаны соответствующие callback-функции, упомянутые выше, для узкого множества функций ограничений. Эти функции имеют вид многочленов третьего порядка:

$$\sum c_i x_{i1} x_{i2} x_{i3} + \sum c_j x_{j1} x_{j2} + \sum c_k x_k$$

Матрицы производных для таких ограничений записываются легко, в отличие от производных для задачи общего вида.

Для базового алгоритма без дополнительных ограничений достаточно такого представления для целевой функции и функций ограничений. Более того, большинство геометрических ограничений могут быть аппроксимированы в таком виде с достаточно хорошей точностью.

7. Дополнительные ограничения

В решаемой мною прикладной задаче очень важна симметрия получаемого решения. Есть несколько групп часто встречаемых дополнительных ограничений, которые стоит затронуть и пояснить как их записать в рамках моего интерфейса.

Первой такой группой являются ограничения большего, чем третий порядок. Данная проблема решается методом введения дополнительных переменных, равных произведению двух или трех других, что позволяет понизить порядок уравнений.

Ко второй группе отнесем ситуации, когда нам нужно соотношение двух переменных "x / y" Для таких ситуаций необходимо завести переменную z, записать уравнение $z*y - x = 0$ и использовать z там, где нужно соотношение. Но если это соотношение используется в неравенствах и y имеет непостоянный знак, то Ipopt может некорректно обработать ситуацию.

Для симметрии объекта часто необходимо, чтобы группа каких-то значений c_i не сильно отличались друг от друга. Чтобы не писать для них, что их разница попарно меньше какой-то величины. Можно ввести переменные min и max и для всех c_i записать следующие уравнения:

$min < c_i$; $max > c_i$; $max - min < dev$

где dev допустимая разница между c_i.

Чаще всего, когда нужны дополнительные ограничения, эти ограничения касаются либо соотношений расстояний, либо соотношений углов. Формула расстояния от точки до плоскости является уравнением второго порядка и с ней проблем не возникает. Для расстояний между точками нужна операция взятия корня, которой мы не обладаем. Это решается двумя методами. Либо можно взять расстояние проекции на какую-то ось, которое считает через скалярное произведение, и работать с проекциями. Либо просто записать уравнения:

$r > 0$

$r^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$

Так как задача решается численными методами, то данные уравнения позволяют использовать r в любых целях, не извлекая корень как таковой вообще.

Для ограничений на углы проще всего использовать косинусы углов, вычисляемые, через скалярное произведение. Но для очень маленьких углов, а они часто встречаются в условиях жесткой симметрии на азимуты каких-то граней, необходимо использовать синусы углов, так как косинус не чувствителен к малым изменениям в районе нуля. Так как

$sin\ x \sim x$,

$cos\ x \sim 1 - x^2$

в районе нуля. Синус вычисляется через векторное произведение, которое также является уравнением третьего порядка.

Все эти, казалось бы, элементарные аппроксимации позволяют работать с достаточно большой для прикладных задач точностью, при этом использую только уравнения третьего порядка, что значительно сокращает время работы алгоритма по сравнению с использованием уравнений высокого порядка, корней и обратных тригонометрических функций.

8. Распределенное тестирование

Развитие алгоритма требует постоянного тестирования на большом количестве примеров и с различными настройками. Чтобы принять решение о введении новой идеи алгоритма, необходимо запустить многочисленные тесты, чтобы выявить слабые места (то есть ситуации, когда новая идея приводит к замедлению алгоритма или ухудшению качества результата). Для решения этой задачи была создана система удалённого тестирования задачи.

На текущий момент система тестирования представляет собой следующее. Для выполнения теста создается его описание в формате json, которое содержит полный набор тестовых данных, на которых будет запускаться пробный алгоритм, перечень всех настроечных параметров, ссылки на действующие реализации алгоритма (dll), а так же дополнительные файлы, если они могут изменяться.

После создание тестового задания оно заносится в очередь. Транспортным каналом передачи данных на сервер является система Dropbox, что оказывается очень удобно, так как позволяет следить за ходом работы с широкого спектра мобильных устройств, например с iPad.

Далее на тестовом сервере по расписанию запускается скрипт проверяющий очередь (queue_starter.py), и в случае если очередь не

пуста и на данный момент сервер свободен, запускает первое задание из очереди.

Запуск осуществляется отдельным скриптом (big_button.py). Этот скрипт по полученному тестовому заданию подготавливает все необходимые данные для запуска. После чего параллельно запускает множество процессов с тестируемым приложением на разных данных. Одновременно выполняются по одному процессу на каждом ядре сервера (или по два, если ядра имеют технологию Hyper threading). В случае падения приложения, скрипт анализирует уже пройденные тесты и перезапускает приложение заново для продолжения задачи. По окончанию работы собирается отчет о результатах в формате json. А далее на его основе формируется удобная для визуального представления результатов таблица в формате Excel.

Помимо основного способа формирования отчетов, есть скрипты, которые создают сравнительные таблицы на основании нескольких отчетов. Общая система генерации этих отчетов report.py содержит мини-язык генерации сводных тестируемых параметров. При необходимости, мы может легко ввести новый параметр, оценивающий работу алгоритма, и вывести его для всех тестов, включая уже пройденные. Например, показать минимальное, максимальное, среднее значение, стандартное отклонение и т.п.

Надо заметить, что язык Python оказывается очень удобен для написания подобных тестовых систем. По сравнению с языком R, который для таких задачи считается более подходящим, Python оказался предпочтительней в связи с гибкими возможностями программирования и решением смежных задач – генерацией отчетов.

Получившая система оказывается достаточно стабильной, устойчивой к падениям тестируемой системы, позволяет глубоко исследовать результаты тестов и находить неочевидные зависимости. Планируется развитие системы тестирования, которое позволит запускать задания в облаке, что значительно сократит время ожидания масштабных тестов.

9. Результаты

Алгоритм был реализован и протестирован на задачах разного размера. Задачи варьировались от простых учебных примеров с простыми, подобными или просто легко вписываемыми многогранниками с количество вершин меньше пятидесяти до реальных прикладных задач с количеством вершин от ста до пятисот и ограничениями на симметрию разной жесткости.

Второй метод на реальных задачах, даже без дополнительных ограничений, работает на порядок быстрее первого, не смотря на увеличенное количество переменных. Исходя из этого и удобства записи уравнений, он был признан более эффективным.

На реальных примерах алгоритм, основанный на втором методе, улучшает любой результат других алгоритмов в исследуемой области на 2-5% объема.

Основные тестируемые примеры содержали вписываемый многогранник с количеством вершин и граней порядка 150 и внешний многогранник с количеством вершин и граней порядка 500, общее количество ограничений порядка 10000. Время построения задачи на этих примерах составляет около одной секунды. Время вычисления на солвере без ограничений на симметрию ~5 секунд. Время вычислений на солвере с ограничениями на симметрию ~30-40 секунд в зависимости от жесткости ограничений. В случаях, когда внешний многогранник был сильно невыпуклый среднее время работы 60 секунд

В своей работе Ipopt использует различные библиотеки для решения систем линейных уравнений и BLAS'ы (библиотеки базовых операций линейной алгебры). Были проведены эксперименты по запуску алгоритма на разных сборках Ipopt, включающих в себя разные комбинации библиотек. Некоторые из этих библиотек могут работать в многопоточном режиме, поэтому отличается еще и количество потоков. Тестирование показало, что наиболее быстрыми оказались сборки, использующие библиотеку MA57.

10. Заключение

В статье рассмотрены два варианта алгоритма вписывания одного многогранника в другой и приведена их сравнительная характеристика. Предложены готовые программные продукты, которые значительно упрощают написание данного алгоритма. Рассмотрены возможные способы оптимизации времени работы алгоритма, варианты его модификаций в зависимости от прикладной задачи, основные идеи упрощения ограничений. Алгоритм протестирован на реальных данных и получены хорошие результаты по качеству и времени работы. Алгоритм является эффективным и актуальным, потому что за счет сведения к НП-

задачи можно распараллелить вычисления, что крайне важно с учетом развития современных вычислительных технологий. Приведена эффективная система тестирования алгоритма в распределенной среде.

11. Список литературы

[1] *A.Tarasov, O.Musin*, The strong thirteen spheres problem,- Topology, Geometry, and Dynamics: Rokhlin Memorial January 11-16, 2010. Saint Petersburg, Russia

[2] *T. Hales*, The status of the Kepler conjecture, Mathematical Intelligencer 16(1994), 47-58.

[3] *В.А. Емеличев, М.М.Ковалев, М.К. Кравцов*, Многогранники, графы, оптимизация, - Москва "Наука", 1981

[4] *R. Fourer, D. M. Gay, and B. W. Kernighan*, AMPL: A Modeling Language For Mathematical Programming, - Duxbury Press/Brooks/Cole Publishing Company, 2003

[5]*Y. Kawajir, C. Laird, A. Wächter*, Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt, 2010, http://www.coin-or.org/Ipopt/documentation/