Обзор по некоммутативной оптимизации*

А.А. Мищенко, А.В. Трейер

14 января 2015 г.

Содержание

1 Введение

Задачи комбинаторной оптимизации являются общепризнанными математическими моделями, описывающими широкий круг актуальных прикладных задач в области исследования операций, таких как задачи об оптимальном распределении заданий, маршрутизации, покрытии, обучении распознаванию образов и т.п. Большинство из этих задач можно переформулировать на языке алгебраических систем, содержащих бинарную

*Работа выполнена при финансовой поддержке гранта РНФ 14-11-00085.

алгебраическую операцию, таких как полугруппы, группы и кольца. Кроме того, основные проблемы комбинаторной оптимизации как теория алгоритмов, сложность соответствующих алгоритмов естественным образом формулируется и для задач для алгебраических систем выше. Этот класс задач и методы решения их в алгебраических системах мы будем называть некоммутативной комбинаторной оптимизацией.

Некоммутативная комбинаторная оптимизация - это только еще зарождающая область исследований, смысл которой - расширить применение классической оптимизации на более широкий класс математических структур, в частности - на классические алгебраические структуры. Работы по некоммутативной комбинаторной оптимизации начаты в ряде университетов США и Европы. В тесном взаимодействии с этими коллективами работает несколько математиков из лаборатории «Информационная безопасность» Омского государственного технического университета и лаборатории комбинаторных и вычислительных методов алгебры и логики Института математики СО РАН. Часть результатов из этих работ мы представляем в данном обзоре, который состоит из двух частей. В первой части обзора мы напоминаем формулировки классических проблем комбинаторной оптимизации и несколько хорошо известных результатов об этих проблемах. Эта часть написана в популярной форме и понятно, что эта часть статьи не обязательна для специалистов по комбинаторной оптимизации. Во второй части мы излагаем основные результаты полученные математиками в Омске и в университетах США и Европы по некоммутативной комбинаторной оптимизации на группах.

2 Некоторые классические проблемы дискретной оптимизации

В этом параграфе мы рассмотрим несколько классических проблем дискретной оптимизации.

2.1 Задача о сумме подмножеств

Задача о сумме подмножеств является важной задачей в теории сложности алгоритмов и криптографии. Задача заключается в нахождении (хотя бы одного) непустого подмножества некоторого набора чисел, чтобы сумма чисел этого подмножества равнялась нулю. Например, пусть задано множество $\{-7, -3, -2, 5, 8\}$, тогда подмножество $\{-3, -2, 5\}$ дает в сумме ноль. Задача является **NP**-полной.

Эквивалентной является задача нахождения подмножества, сумма элементов которого равна некоторому заданному числу s. Задачу о сумме подмножеств также можно рассматривать как некоторый специальный случай задачи о рюкзаке (смотри параграф 2.2). Интересным случаем задачи о суммировании подмножеств является задача о разбиении, в которой s равна половине суммы всех элементов множества.

Сложность. Вычислительная сложность задачи о сумме подмножеств зависит от двух параметров – числа N элементов множества, и точности P (определяется как число двоичных разрядов в числах, составляющих множество).

Сложность наилучшего известного алгоритма экспоненциальна по наименьшему из двух параметров N и P. Таким образом, задача наиболее трудна, когда N и P имеют один порядок. Задача становится лёгкой только при очень маленьких N или P.

Если N (число переменных) мало, то полный перебор вполне приемлем. Если P (число разрядов в числах множества) мало, можно использовать динамическое программирование.

Экспоненциальный алгоритм. Имеется несколько путей решения задачи за время, экспоненциально зависящее от N. Наиболее простой алгоритм просматривает все подмножества и, для каждого из них, проверяет, является ли сумма чисел подмножества приемлемой. Время работы алгоритма оценивается как $O(N2^N)$, поскольку имеется 2^N подмножеств, а для проверки каждого подмножества нам нужно сложить не более N элементов.

Лучше алгоритм со временем работы $O(2^{N/2})$. Этот алгоритм делит все множество из N элементов на два подмножества по $N/2$ элементов в каждом. Для каждого из этих подмножеств строится набор сумм всех $2^{N/2}$ возможных подмножеств. Оба списка сортируются. Если использовать простое сравнение для сортировки, получим время $O(N2^{N/2})$. Для сортировки мы можем применить технику слияния. Имея сумму для k элементов, добавим $k+1$-ый элемент и получим два сортированных списка, затем совершим слияние списков (без добавленного элемента и с добавленным элементом). Теперь мы имеем список сумм для $(k+1)$ элементов, и для его создания потребовалось $O(2^k)$ времени. Таким образом, каждый список может быть создан за время $O(2^{N/2})$. Имея два сортированных списка, алгоритм может проверить, могут ли дать суммы элементов из первого и второго списка значение s за время $O(2^{N/2})$. Для этого алгоритм просматривает первый список в порядке убывания (начиная с самого большого элемента), а второй список в порядке возрастания (начиная с наименьшего элемента). Если сумма текущих элементов больше s, алгоритм передвигается к следующему элементу в первом списке, а если меньше s, к следующему элементу во втором списке. Если же сумма равна s, решение найдено и алгоритм останавливается. Horowitz и Sahni впервые опубликовали этот алгоритм в 1974 году [2].

Приближенный алгоритм, работающий за полиномиальное время. Существует версия приближенного алгоритма, дающего для заданного множества из N элементов $x_1, x_2, \ldots, x_N$ и числа s следующий результат:

- **ДА**, если существует подмножество с суммой элементов s;
- **НЕТ**, если нет подмножества, имеющего сумму элементов между $(1-c)s$ и s для некоторого малого $c > 0$;
- Любой ответ (**ДА** или **НЕТ**), если существует подмножество с суммой элементов между $(1-c)s$ и s, но эта сумма не равна s.

Если все элементы неотрицательны, алгоритм дает решение за поли-

номиальное от N и $1/c$ время.

Алгоритм обеспечивает решение исходной задачи нахождения суммы подмножеств в случае, если числа малы (опять же, неотрицательны). Если любая сумма чисел имеет не более P бит, то решение задачу для $c = 2^{-P}$ эквивалентно нахождению точного решения. Таким образом, полиномиальный приближенный алгоритм становится точным со временем выполнения, зависящим полиномиально от N и 2^P (то есть, экспоненциально от P).

Алгоритм приближенного решения задачи о сумме множеств работает следующим образом:

- Создаем список S, первоначально состоящий из одного элемента 0.
- Для всех i от 1 до N выполним следующие действия:
  - Создаем список T, состоящий из $x_i + y$ для всех y из S;
  - Создаем список U, равный объединению T и S;
  - Сортируем U;
  - Опустошаем S;
  - Пусть y – наименьший элемент U;
  - Внесем y в S;
  - Для всех элементов z из U, перебирая их в порядке возрастания выполним:

    Если $y + cs/N < z \leq s$, положим $y = z$ и внесем z в список S (тем самым мы исключаем близкие значения и отбрасываем значения, превосходящие s).
- Если S содержит число между $(1-c)s$ и s, говорим **ДА**, в противном случае – **НЕТ**.

Алгоритм имеет полиномиальное время работы, поскольку размер списков S, T и U всегда полиномиально зависим от N и $1/c$ и, следовательно, все операции над ними будут выполняться за полиномиальное время.

Сохранить размер списков полиномиальным позволяет шаг исключения близких значений, на котором мы добавляем элемент z в список S только если он больше предыдущего на cs/N и не больше s, что обеспечивает нам включения не более N/c элементов в список.

Алгоритм корректен, поскольку каждый шаг дает суммарную ошибку не более cs/N и N шагов вместе дадут ошибку, не превосходящую cs.

2.2 Задача о рюкзаке

Задача о рюкзаке (Knapsack problem) – одна из задач комбинаторной оптимизации, принадлежащая классу **NP**. Название своё получила от максимизационной задачи укладки как можно большего числа ценных вещей в рюкзак при условии, что общий объём (или вес) всех предметов, способных поместиться в рюкзак, ограничен. Задачи о рюкзаке и её модификации часто возникают в экономике, прикладной математике, криптографии, генетике и логистике для нахождения оптимальной загрузки транспорта (самолёта, поезда, трюма корабля) или склада. В общем виде задачу можно сформулировать так: из заданного множества предметов со свойствами «стоимость» и «вес», требуется отобрать некое число предметов таким образом, чтобы получить максимальную суммарную стоимость при одновременном соблюдении ограничения на суммарный вес.

Классическая постановка задачи. Пусть имеется набор из n предметов, для каждого предмета определён вес $w_i > 0$ и ценность $v_i > 0$, $i = 1, 2, \ldots, n$. Дана грузоподъёмность W. Выбрать подмножество предметов, так чтобы их общий вес не превышал W, а суммарная их ценность была бы максимальной.

Существует множество разновидностей задачи о рюкзаке, отличия заключаются в условиях, наложенных на рюкзак, предметы или их выбор:

- Рюкзак 0-1 (0-1 Knapsack Problem): не более одного экземпляра каждого предмета;
- Ограниченный рюкзак (Bounded Knapsack Problem): не более заданного числа экземпляров каждого предмета;

- Неограниченный рюкзак (целочисленный рюкзак) (Unbounded Knapsack Problem (integer knapsack)): произвольное количество экземпляров каждого предмета;
- Рюкзак с мультивыбором (Multiple-choice Knapsack Problem);
- Мультипликативный рюкзак (Multiple Knapsack Problem): есть несколько рюказков, каждый со своим максимальным весом. Каждый предмет можно положить в любой рюкзак или оставить;
- Многомерный рюкзак (Multy-dimensional knapsack problem): вместо веса дано несколько разных ресурсов (например, вес, объём и время укладки). Каждый предмет тратит заданое количество каждого ресурса. Надо выбрать подмножество предметов так, чтобы общие затраты каждого ресурса не превышали максимума по этому ресурсу, и при этом общая ценность предметов была максимальна.

Наглядная постановка задачи о рюкзаке привела к тому, что она нашла применение в разных областях знаний: в математике, информатике и на стыке этих наук – в криптографии.

Доподлинно неизвестно, кто первым привел математическую формулировку задачи о рюкзаке. Одно из первых упоминаний о ней можно найти в статье Джорджа Балларда Мэтьюса [3], датированной 1897 годом. Интенсивное изучение данной проблемы началось после публикации Д. Б Данцигом в 1957 году книги «Discrete Variable Extremum Problem» [4], особенно в 70-90-е годы 20-го века, как теоретиками, так и практиками. Во многом данный интерес вызван достаточно простой формулировкой задачи, большим числом её разновидностей и свойств и в то же время сложностью их решения. В 1972 году данная задача вошла в список К. Мэннига **NP**-полных задач [5].

С практической точки зрения задача о рюкзаке может служить моделью для большого числа промышленных ситуаций:

- Размещение грузов в помещении минимального объёма;

- Раскройка ткани – для заданного куска материала получить максимальное число выкроек определенной формы;
- Расчет оптимальных капиталовложений;
- С задачей о рюкзаке сталкивается любой человек, собирающий рюкзак.

Сложность. Задача о рюкзаке относится к классу **NP**-полных задач, для неё нет полиномиального алгоритма, решающего её за разумное время. Поэтому при решении задачи о рюкзаке всегда нужно выбирать между точными алгоритмами, которые не применимы для «больших» рюкзаков, и приближенными, которые работают быстро, но не обеспечивают оптимального решения задачи. Естественно, создание быстрого и достаточно точного алгоритма представляет большой интерес.

Точные алгоритмы. К точным алгоритмам относятся алгоритм полного перебора и метод ветвей и границ.

- **Полный перебор.** Пусть в рюкзак загружаются предметы N разных типов. Рассмотрим задачу, когда количество предметов каждого типа не ограничено. Нужно определить максимальную стоимость груза, вес которого равен P. Для получения решения алгоритмом полного перебора осуществляется перебор всех вариантов загрузки рюкзака. Временная сложность алгоритма $O(N!)$, то есть он работоспособен для небольших значений N. С ростом N задача становится неразрешимой данным методом за приемлемое время.

- **Метод ветвей и границ.** Метод ветвей и границ является вариацией метода полного перебора с той разницей, что мы сразу исключаем заведомо неоптимальные решения.

  Пусть есть оптимальное решение R. Попытаемся его улучшить, рассмотрев решение на другой ветви. Если на рассматриваемой в данной момент ветви решение становится хуже (с какого-то шага), чем R, то прекращаем его исследование и выбираем другую ветвь дерева. Однако метод ветвей и границ работает не для всех наборов данных.

Можно привести примеры, в которых время выполнения будет таким же, как и для простого перебора.

Приближённые алгоритмы. К приближенным алгоритмам относятся следующие:

- **Жадный алгоритм.** Согласно жадному алгоритму предметы сортируются по убыванию стоимости единицы каждого. Помещаем в рюкзак то, что помещается и одновременно и самое дорогое, то есть с максимальным отношением цены к весу. Для сортировки предметов потребуется $O(N\log(N))$. Далее организуется проход по всем N элементам цикла. Точное решение можно получить не всегда, поэтому жадный алгоритм относится к приближенным алгоритмам.

- **Генетический алгоритм**. Генетические алгоритмы были предложены Джоном Генри Холландом в 1970 году и относятся к так называемым метаалгоритмам. Идея – составление алгоритмов поиска на основе биологической модели механизмов естественного отбора. Базовыми понятиями являются: популяция, отбор, мутация, скрещивание.

  Популяция. Составляется набор бинарных строк (хромосом), возможных решений. На основе первой («старой») популяции строится вторая («новая») популяция решений, которая служит «старой» для третьей популяции и т.д.

  Отбор. Задается функция выбора, согласно которой, лучшие представители «старой» популяции выбираются для воспроизводства «новой». Следовательно, алгоритм выбирает наилучшее решение.

  Скрещивание. Для пары строк («родителей») с определенной длиной r выбирается произвольное число $1 \leq s \leq r$. «Родители» обмениваются между собой битами с $s+1$-го по r-й и получаются две новые строки («потомки»).

  Мутация. Изменение, происходящее с определенной вероятностью.

Содержимое рюкзака представляется в виде хромосом или бинарных строк, i-й бит которых равен единице в случае наличия предмета в рюкзаке, нулю – в случае его отсутствия. Задается целевая функция S – вместимость рюкзака.

Алгоритм имеет следующие этапы:

1. Создание случайной популяции двоичных хромосом.
2. Отбор
3. Скрещивание
4. Мутация
5. Новая популяция
6. Проверка достигнут ли результат, если нет, то переход на шаг 2.

Алгоритм прерывается после заданного числа итераций.

Генетический алгоритм не гарантирует нахождение оптимального решения, однако показывает хорошие результаты за меньшее время по сравнению с другими алгоритмами.

3 Некоммутативные группы

Пусть дана группа G, заданная с помощью порождающего множества X и множества определяющих соотношений R.

$$G = \langle X |\ R \rangle.$$

В данном параграфе, следуя статье А. Мясникова, А. Николаева и А. Ушакова [6], мы введем задачи оптимизации для группы G, которые являются аналогами классических проблем дискретной оптимизации.

Постановка задачи о сумме подмножеств (subset sum problem).

The subset sum problem SSP(G, X)**:** *Даны элементы* $g_1, \ldots, g_k, g \in G$, *определить выполняется ли следующие равенство в группе* G*:*

$$g = g_1^{\varepsilon_1} \ldots g_k^{\varepsilon_k} \tag{1}$$

для некоторых $\varepsilon_1, \ldots, \varepsilon_k \in \{0, 1\}$.

В [6] показано, что задача $\mathbf{SSP}(G)$ является **NP**-полной для многих хорошо известных групп G, которые обычно считаются хорошими для разного рода вычислительных задач. А именно, задача $\mathbf{SSP}(G)$ является **NP**-полной для свободных метабелевых групп конечного ранга $r \geq 2$, для сплетения групп $\mathbb{Z} \wr \mathbb{Z}$ или в более общем случае – сплетение любых двух конечно порожденных бесконечных абелевых групп. Данные группы конечно порождены, но не конечно представлены. Более того, оказалось, что задача $\mathbf{SSP}(G)$ является **NP**-трудной для всех групп Баумслага-Солитера $BS(1, p)$ для $p \geq 2$. Напомним, что группа Баумслага-Солитера задается следующим представлением:

$$BS(m, n) = \langle a, t |\ t^{-1} a^m t = a^n \rangle,$$

где $m, n \in \mathbb{Z}$.

Более того, не трудно проверить, что задача $\mathbf{SSP}(G)$ является **NP**-трудной если она **NP**-трудна для некоторой подгруппы из G. Таким образом, $\mathbf{SSP}(G)$ будет **NP**-трудной для любой группы G, которая содержит подгруппу, изоморфную приведенным выше группам.

Также известны примеры для которых задача $\mathbf{SSP}(G)$ решается хорошо, а именно, в [6] показано, что задача $\mathbf{SSP}(G)$ является полиномиальной для любой конечно порожденной нильпотентной группы G и для любой гиперболической группы G.

The knapsack problem $\mathbf{KP}(G, X)$: *Даны элементы* $g_1, \ldots, g_k, g \in G$, *определить выполняется ли следующие равенство в группе* G*:*

$$g = g_1^{\varepsilon_1} \ldots g_k^{\varepsilon_k} \tag{2}$$

для некоторых не отрицательных целых чисел $\varepsilon_1, \ldots, \varepsilon_k$.

Наряду с задачей $\mathbf{KP}(G, X)$ можно рассматривать ограниченную версию данной задачи $\mathbf{BKP}(G, X)$ (bounded knapsack problem).

Bounded knapsack problem $\mathbf{BKP}(G, X)$: *Даны элементы* $g_1, \ldots, g_k, g \in G$ *и число* $1^m \in \mathbb{N}$, *определить выполняется ли следующие равенство в*

группе G*:*

$$g = g_1^{\varepsilon_1} \dots g_k^{\varepsilon_k}$$

для некоторых $\varepsilon_i \in \{0, 1, \dots, m\}$.

Задача $\mathbf{BKP}(G)$ за полиномиальное время сводится к задаче $\mathbf{SSP}(G)$ и обратно, поэтому для всех групп G, указанных выше, для которых известна сложность $\mathbf{SSP}(G)$, задача $\mathbf{BKP}(G)$ будет иметь такую же сложность.

Также в работе [6] для гиперболических групп построено полиномиальное свидение задачи $\mathbf{KP}(G)$ к задаче $\mathbf{BKP}(G)$, которая в свою очередь для гиперболических групп решается за полиномиальное время.

В работе [7] показано, что для нильпотентных групп ранга $r \geq 2$ задача $\mathbf{KP}(G)$ алгоритмически не разрешима, так как к ней можно свести любую систему квадратный уравнений над целыми числами. Это является очень интересным результатом, так как задача $\mathbf{SSP}(G)$ и $\mathbf{BKP}(G)$ для нильпотентных групп решается за полиномиальное время. Этот пример является одним из двух известных нам примеров, когда алгоритмическая задача для группы G не разрешима, а ограниченная версия этой же задачи решается за полиномиальное время на группе G.

4 Открытые проблемы

Проблема $\mathbf{SSP}(G)$ и $\mathbf{BKP}(G)$ всегда разрешимы в группах где разрешима проблема равенства слов. На данный момент остаются открытыми вопросы сложности данных задач для полициклических групп и для группы фонарщика (lamplighter group, то есть группа $\mathbb{Z}_2 \wr \mathbb{Z}$).

Если для двух групп A и B известна сложность задачи $\mathbf{SSP}$, то что можно сказать про сложность задачи $\mathbf{SSP}(G)$, если группа G представима в виде свободного произведения с объединением групп A и B? Так же интересен вопрос про сложность задачи $\mathbf{SSP}(G)$, если группа G является HNN-расширением группы A?

Список литературы

[1] Wikipedia.

[2] E. Horowitz, S. Sahni, *Computing partitions with applications to the knapsack problem*, Journal of the Association for Computing Machinery, (1974), T. 21: 277–292.

[3] G. B. Mathews, *On the partition of numbers*, Proc. London Math. Soc. (1896) s1-28 (1): 486-490.

[4] George B. Dantzig, *Operations Research*, Vol. 5, No. 2 (Apr., 1957), pp. 266-277.

[5] Richard M. Karp, *Reducibility among Combinatorial Problems*, Complexity of Computer Computations, The IBM Research Symposia Series (1972), pp 85-103.

[6] A. Myasnikov, A. Nikolaev, A. Ushakov, *Knapsack Problems in Groups* // http://arxiv.org/abs/1302.5671

[7] A. Mishchenko, A. Treyer, *Knapsack Problems in Nilpoten Groups* // preparing for publication.