Санкт-Петербургский государственный политехнический университет
Физико-механический факультет
Кафедра прикладная математика

Диссертация допущена к защите
Зав. кафедрой
_____ **В.Е. Клавдиев**

«____»_____

ДИССЕРТАЦИЯ
на соискание степени МАГИСТРА

Тема: ***Схемы вычисления полиномов и их приложения***

Направление: 010500 – Прикладная математика и информатика
Магистерская программа: 510209 –
Математическое и программное обеспечение компьютерных систем

Выполнил студент гр. 6057/2 М.П. Кожевников

Руководитель, к.ф.-м.н., с.н.с. Н.Н. Васильев

Консультанты:
по охране труда, к.т.н., доц. В.В. Монашков

Санкт-Петербург
2010

Содержание

Введение

Предметом данной работы являются различные схемы вычисления полиномов и их приложения к

- эффективному *вычислению полиномов с предобработкой* без обработки коэффициентов
- редукция[1] систем полиномов относительно полиномиального базиса методом *инкрементальной редукции (МИР)*

Целью работы является сравнительный анализ различных подходов к построению таких схем в контексте указанных приложений, а также их эффективная реализация.

Синтез вычисляющих программ

Вычисление полиномов с предобработкой обычно относят к *синтезу вычисляющих программ*, несмотря на то, что, зачастую, программой результат предобработки можно назвать лишь в довольно общем смысле – как правило, это некоторая форма описания последовательности действий, которые необходимо произвести над значениями переменных с тем, чтобы получить значение полинома от этих переменных. Разумеется, такая последовательность может быть записана в форме программы на интерпретируемом языке или, чаще, исполняемого модуля компилируемого языка, откуда и происходит термин.

Синтез вычисляющих программ как подход к вычислению значений функций сформировался достаточно давно – в конце пятидесятых годов прошлого века. Его задача заключается в построении эффективной процедуры вычисления значения заданной функции, что было вдвойне актуально с учетом мощности ЭВМ того времени. В большинстве случаев вычисление значения искомой функции так или иначе сводилось к вычислению значения некоторого полинома – например, через разложение в ряд или интерполяцию.

Несмотря на интенсивный рост производительности процессоров, задача актуальна и сейчас – для некоторых алгоритмов симуляции или численного решения дифференциальных уравнений, а также в тех случаях, где вычисления производятся с высокой точностью и, как следствие, существенно возрастает стоимость каждой арифметической операции.

В данной работе мы рассматриваем схемы вычисления полиномов многих переменных без обработки коэффициентов – эта область несколько менее изучена по сравнению с полиномами одной переменной. Данная задача рассматривалась и ранее, например, в [7, 8]. Нас же интересует сравнительный анализ различных подходов к построению схем в терминах их сложности и практической эффективности.

[1] В данном контексте можно было бы использовать термин «нормализация», однако он является не вполне корректным, так как относительно произвольного полиномиального базиса полином может не иметь нормальной формы

Редукция

Схема вычисления полинома может применяться не только для вычисления его значений, но и для выполнения произвольной операции подстановки (см. 1.2.1). В частности, схему можно использовать для более эффективной редукции полинома или системы полиномов относительно полиномиального базиса (см. 3). Последняя же задача представляет интерес как сама по себе, так и в качестве компонента для алгоритма вычисления базиса Грёбнера [1], чья эффективность во много определяется эффективностью алгоритма редукции [5]. Совместная редукция нескольких полиномов может также быть полезна в ряде алгоритмов [20].

Задачи

Изначально мы рассчитывали, что данная работа будет преимущественно ориентирована на создание эффективного символьно-численного интерфейса для различных типов полиномов, такого, как `СПРИНТ` [8]. Однако, в процессе работы мы обнаружили, что обоснование актуальности такой системы и ее сравнение с существующими аналогами представляет определенную сложность, так как

- ни один из известных нам программных пакетов общего назначения не содержит специальных средств для многократного вычисления полиномов с предобработкой, и, следовательно, сравнение с этими пакетами было бы не вполне «честным»
- количество практических задач, где возникают достаточно сложные полиномы высоких степеней весьма ограничено
- для тех задач, где это требуется, обычно разрабатываются специальные системы с учетом особенностей задачи, что существенно осложняет сравнительный анализ

Кроме того, данная тема достаточно хорошо изучена и на сегодняшний день существенно новых результатов логично ожидать лишь для задач, где возникают полиномы специального вида, либо в области параллельных вычислений.

В свете вышесказанного было решено не делать сравнить потенциальную эффективность рассматриваемых схем лишь между собой, а основной темой работы сделать методы редукции полиномов на основании схем подстановки. Первоначальная тема, однако, наложила свой отпечаток на используемые средства разработки – была создана небольшая библиотека для работы с полиномами над конечными полями или кольцом рациональных чисел на `C++`, с использованием `GMP` [24].

Для работы с алгоритмами редукции, где производительность определяется в первую очередь качеством используемых алгоритмов, а не эффективностью элементарных операций, было бы лучше использовать один из существующих пакетов компьютерной алгебры

с открытым программным интерфейсом, например, `Singular` [23] или `Axiom` [22]. К таким пакетам существуют также интерфейсы [25] на языках более высокого уровня, что обеспечивает большую гибкость при постановке вычислительных экспериментов.

1 Постановка задачи

Мы будем рассматривать преимущественно полиномы с целыми или рациональными коэффициентами, либо с коэффициентами в некотором конечном поле. Бо̀льшая часть утверждений верна для любого типа коэффициентов. В тех случаях, когда это не так, тип коэффициентов будет оговорен отдельно.

1.1 Основные определения

Введем основные понятия, которые потребуются нам в данной работе.

Определение 1. *Мономом называется произведение неотрицательных целых степеней заданных переменных* $\prod_i x_i^{n_i}$. *Множество всевозможных мономов на заданном наборе переменных образует моноид и обозначается* $\mathbb{M}$. *Сумма показателей степеней* $\sum_i n_i$ *называется полной степенью монома.*

Определение 2. *Линейное пространство над полем* $\mathbb{K}$ *с базисом* $\mathbb{M}$ *образует кольцо полиномов* $\mathbb{K}[x]$. *Членом полинома называется произведение коэффициента и соответствующего монома. Множество базисных элементов, имеющих в полиноме* $p \in \mathbb{K}[x]$ *ненулевые коэффициенты, называется его носителем* $s(p)$.

Определение 3. *Линейное отношение порядка* $>$ *на множестве* $\mathbb{M}$ *называется мономиальным упорядочением. Если, кроме того,* $(\forall u \in \mathbb{M}\, u > 1) \wedge (\forall u, v, w \in \mathbb{M}\, u > v \Rightarrow wu > wv)$, *то упорядочение называется допустимым.*

1.2 Схемы вычисления полиномов

Схемы вычисления полиномов можно рассматривать как алгоритм вычисления значения полинома, записанный в терминах умножения и сложения полиномов, а также умножения полинома на число (коэффициент). В некоторых случаях возведение в степень также рассматривают а качестве элементарной операции.

Такие схемы удобно изображать в виде диаграмм (рис. 1), где корневые узлы отвечают переменным и единице, а промежуточные соответствуют операциям. Входящие дуги при этом указывают источники операндов.

Для большинства полиномов схема вычисления, очевидно, неуникальна. Например, обе схемы на диаграмме 2 соответствуют полиному $xy + xz$. Может возникнуть впечатление, что каждая схема однозначно соответствует некоторой расстановке скобок – на диаграмме 1 это $xy + xz$ слева и $x(y + z)$ справа. Это не совсем верно, так как, например, расстановка скобок не указывает, являются ли два вхождения одного и того же монома ссылками на один узел схемы или на два различных узла. Например, диаграмма 3 изображает две схемы вычисления полиномов, соответствующие (в смысле расстановки скобок) выражению $xy(xy + 1)$.

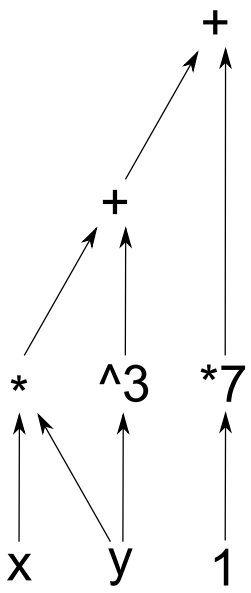

Рис. 1: Две схемы для полинома $xy + xz$

1.2.1 Схема как алгоритм подстановки

В более широком смысле, схема вычисления полинома является схемой подстановки значений в этот полином, причем в качестве значений могут выступать элементы любого кольца, допускающие умножение на коэффициенты рассматриваемого полинома. В частности, в зависимости от типа коэффициентов это могут быть комплексные, вещественные, рациональные или целые числа, либо целые числа по заданному модулю. Кроме того, очевидно, это могут быть полиномы того же типа, что и рассматриваемый.

Таким образом, метки «+» и «*» в узлах схемы соответствуют умножению и сложению в соответствующем кольце. Следует обратить внимание, что умножение на коэффициент не обязательно совпадает с умножением в кольце. В качестве примера можно рассмотреть подстановку векторов с покомпонентным умножением и сложением в полином с целыми коэффициентами.

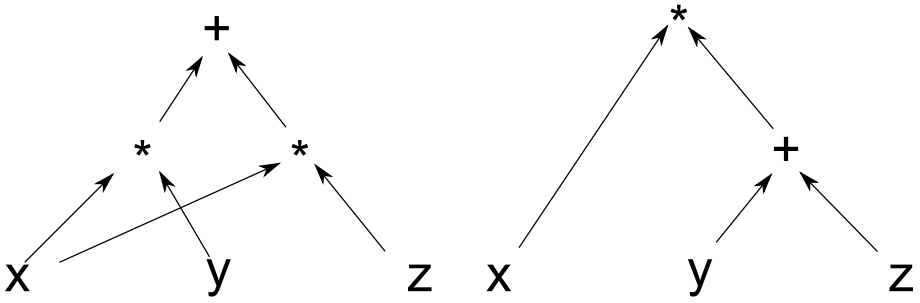

Рис. 2: Две схемы для полинома $xy + xz$

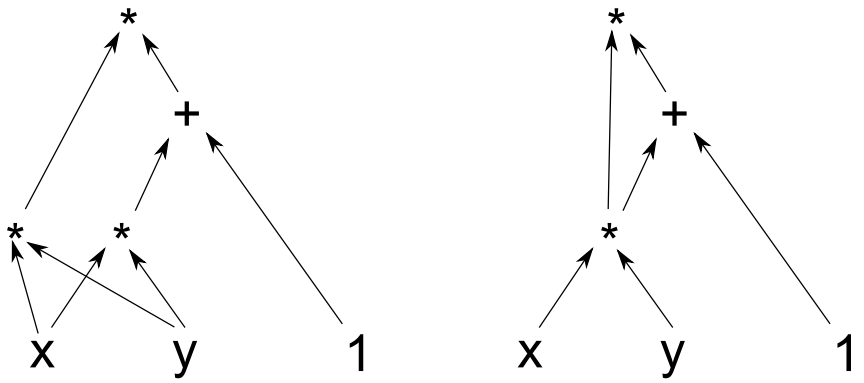

Рис. 3: Две схемы для полинома $xy(xy+1)$

1.2.2 Критерии качества

Нас интересуют схемы, которые позволяют решать определенные задачи более эффективно. Следовательно, естественным критерием качества, как правило, является скорость решения соответствующей задачи. Однако, использование такого критерия на практике часто оказывается неудобным (в силу отсутствия эффективного способа вычисления), поэтому используются различные упрощенные критерии. Для задачи вычисления в качестве упрощенного критерия может выступать общее количество операций, возможно, с весами.

Выбор подходящего критерия качества представляет собой чрезвычайно важный фактор при разработке алгоритма построения схем вычисления полиномов. Критерии, используемые для других приложений, мы обсудим позже.

1.2.3 Полиномы одной переменной

Для случая полиномов одной переменной существует два основных способа построения схем. Первый – очевидный – способ состоит в последовательном вычислении всех мономов и сложении результатов, а второй известен как «схема Горнера». Следует обратить внимание, что семантика слова «схема» в данном названии отлична от используемой в нашем контексте, поэтому здесь и далее мы будем говорить о *методе* Горнера.

Для многих задач метод Горнера дает схему, близкую к оптимальной, однако в ряде случаев существуют и более эффективные подходы [3].

1.2.4 Полиномы многих переменных

В случае полиномов многих переменных построение схемы, оптимальной в смысле заданного критерия качества, является нетривиальной проблемой. Для полиномов высоких степеней построение такой схемы может быть слишком сложным с вычислительной точки зрения.

В большинстве случаев можно эффективно построить схемы, близкие к оптимальной. Это оправдано также и тем, что используемый приближенный критерий качества может не

вполне соответствовать истинному критерию, и, как следствие, оптимальная схема может не быть наиболее эффективной.

1.2.5 Схемы для нескольких полиномов

Понятие схемы естественным образом обобщается на случай совместного вычисления нескольких полиномов. Это актуально преимущественно в задачах вычисления значений полиномов, однако совместная редукция набора многочленов относительно заданного базиса также может иметь свои приложения – например, такая задача возникает при реализации алгоритма F4 вычисления базиса Грёбнера [20], хотя она и формулируется в матричной форме.

1.3 Вычисление значений полиномов

Задача вычисления значений полиномов традиционно пользуется меньшим вниманием по сравнению, например, с задачей умножения полиномов, так как является вычислительно менее трудоемкой. В большинстве случаев схема Горнера оказывается достаточно эффективной. Однако, в тех случаях, когда значение полинома требуется вычислять многократно, выгодно подготовить схему вычисления полинома заранее.

В тех случаях, когда речь идет о вычислении значений полинома одной переменной, часто применяются *схемы с обработкой коэффициентов*. Описание таких схем можно найти, например, в [2, 18, 19]. Так как нас интересуют преимущественно полиномы многих переменных, мы будем заниматься лишь схемами без обработки коэффициентов.

Так как сравнение с аналогичными системами представляет определенную сложность, мы будем рассматривать лишь сравнительную эффективность различных схем.

1.4 Редукция полиномов

Алгоритм МИР, который мы рассмотрим подробнее в разделе 3, можно рассматривать как операцию замены переменных в рассматриваемом полиноме, выполняемую в факторалгебре. Это означает, что результатом умножения мономов является *нормальная форма* их произведения. Не зная значений аргументов, сложность данной операции нельзя определить с какой-либо фиксированной точностью – в практических задачах от операции к операции она может варьироваться на несколько порядков 3.2.1.

Кроме того, в задаче вычисления значений схема используется многократно с различными значениями аргументов, тогда как необходимость редукции одного и того же многочлена относительно многих различных наборов полиномов возникает крайне редко. Как следствие, в первом случае оправдано построение наилучшей возможной схемы, а во втором необходимо соблюдать баланс между качеством схемы и временем ее построения.

Поэтому, хотя обе рассматриваемых нами задачи могут быть выражены в терминах

схем вычисления полиномов, природа их весьма различна, и, как следствие, для них лучше приспособлены различные методы построения схем.

2 Методы построения схем

2.1 Структура схемы

Существуют методы, которые строят упрощенные, в некотором смысле, СВП. В корневых узлах такой схемы стоят не первые степени переменных, а некоторые нетривиальные мономы, которые, в свою очередь, могут быть вычислены различными способами. Соответственно, для них можно построить отдельную схему, называемую схемой *совместного вычисления мономов* (ССВМ).

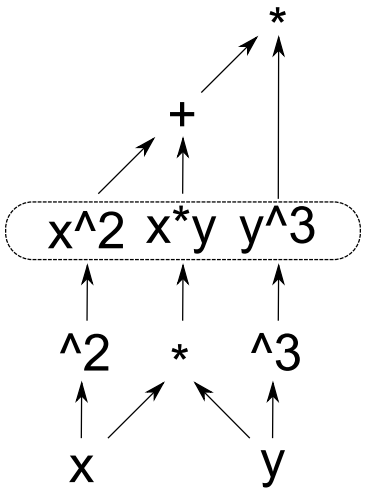

Рис. 4: Схема для $x^2y^3 + xy^4$ как композиция двух схем. Пунктиром выделен переход.

Можно рассматривать это как разбиение схемы на две части (рис. 4). Формально они отличаются лишь тем, что нижняя не содержит операций сложения. Методы построения ССВМ мы рассмотрим в разделе 2.6.

В данной работе мы рассматриваем методы, сохраняющие количество сложений и все операции умножения на коэффициенты. Более того, во всех методах, кроме метода Горнера, домножение на коэффициенты и суммирование производятся в конце. Следовательно, в этих случаях мы можем отбросить эти операции и рассматривать задачу вычисления набора одночленов, то есть задачу построения ССВМ.

Мы будем обозначать носитель рассматриваемого полинома как M, а множество его переменных как V.

2.2 Сложность построения оптимальной схемы

Пусть задан критерий качества $f : s \to h, s \in S, h \in \mathbb{R}$, где S – множество возможных схем. Необходимо для заданного полинома $p(\bar{x}) = \sum_i c_i \bar{x}^{\bar{\alpha}_i}$ построить отвечающую ему схему с наибольшим возможным h.

В разд. 2.6 мы рассмотрим задачу о построении кратчайшей аддитивной последовательности, которая NP-полна и эквивалентна задаче построения оптимальный СВП для

вычисления значений полинома в числах фиксированной длины с плавающей точкой. Нам неизвестно формального доказательства NP-полноты для построения оптимальной СВП для задачи редукции, однако интуитивно можно предполагать, что усложнение критерия качества не приведет к уменьшению сложности построения схемы.

2.3 Тривиальная схема

Существует схема, которая не требует построения – она зафиксирована в форме записи полинома. Согласно этой схемы, мы должны сначала вычислить все мономы, встречающиеся в полиноме, а затем просуммировать результаты. Мы будем называть эту схему тривиальной.

2.4 Обобщенный метод Горнера

Обобщенный (на случай полиномов многих переменных) метод Горнера (ОМГ) прост в реализации и позволяет получать достаточно хорошие схемы. Алгоритм построения 1 является рекурсивным. Фактически, он представляет исходный полином t в виде $t = pd + q$, причем ни один член q не делится на d. Затем операция повторяется для p и q, до тех пор, пока на месте t не окажется одночленом.

Легко видеть, что число рекурсивных вызовов не превышает число членов в исходном полиноме. Сложность каждого вызова составляет $O(n)$ операций вычисления наибольшего общего делителя двух одночленов GCD. Всего не более $O(n^2)$ операций GCD. Сложность одной такой операции можно считать постоянной, если ограничить величину коэффициентов некоторой константой и зафиксировать число переменных.

Например, для полинома $x^2yz + xyz + x^3 + xyz + yz^2 + z^3 + yz + y$ метод Горнера даст схему, соответствующую $x(yz(x + z + 1) + x * (x + 1)) + z(y(z + 1) + z^2) + y$, что дает 7 умножений и 8 сложений для задачи вычисления значений, вместо 17 умножений и 8 сложений, если использовать тривиальную схему.

2.5 Метод минимального покрывающего дерева

Рассмотрим орграф по отношению делимости на множестве $M' = M \cup V$ с весами на ребрах, равными некоторой функции частного, например, его полной степени. Нам необходимо выбрать некоторое подмножество ребер этого графа, которое будет отвечать схеме вычисления рассматриваемого набора мономов. Подмножество должно быть таким, чтобы каждый узел из $M \setminus V$ имел ровно одно входящее ребро, с тем чтобы не вычислять один и тот же моном дважды. Заметим, что такое подмножество всегда существует, так как каждый элемент $M \setminus V$ делится на хотя бы один элемент V. Пример построения можно видеть на рисунке 5. Здесь для краткости опущены все дуги, кроме соединяющих мономы на соседних уровнях.

Algorithm 1: HORNER(T,S)

Input: список T членов полинома в порядке убывания, контейнер схемы S

$P \leftarrow \{\}$

$d \leftarrow T[0]$

if *Length(T)*$= 1$ **then**

  return $AddNode(S, T[0])$

end

foreach $t \in T$ **do**

  $g \leftarrow GCD(t, d)$

  if $TotalDegree(g) > 0$ **then**

    $P \leftarrow P \cup \{t\}$

    $d \leftarrow g$

  end

end

$Q \leftarrow T \setminus P$

$P' \leftarrow \{\}$

foreach $t \in P$ **do**

  $P' \leftarrow P' \cup \{t/d\}$

end

$N_p \leftarrow Horner(P', S)$ `// recursive call`

$N_d \leftarrow AddNode(S, d)$

$N_p \leftarrow AddNode(S, \langle *, N_d, N_p \rangle)$

if $Q \neq \emptyset$ **then**

  $N_q \leftarrow Horner(Q, S)$

  $N_p \leftarrow AddNode(S, \langle +, N_p, N_q \rangle)$

end

return p

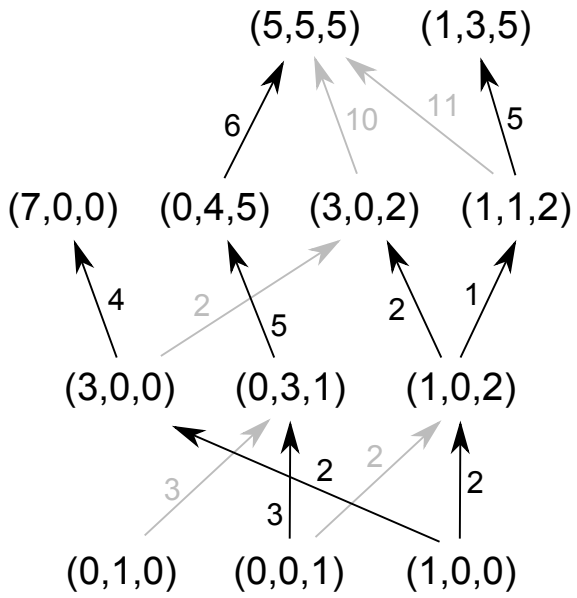

Рис. 5: Граф делимости (неполный). Черные дуги порождают схему.

Среди всех возможных подмножеств, отвечающих данным критериям, мы выберем одно, обладающее наименьшим весом, и обозначим соответствующий подграф G_{min}. Заметим, что название метода не отвечает структуре, которую мы построили – последняя представляет собой не дерево, а лес из $|V|$ деревьев, причем каждое из деревьев может не являться минимальным покрывающим деревом соответствующего подграфа, в силу ограничения на количество входящих ребер. Тем не менее, аналогия с задачей о минимальном покрывающем дереве кажется нам достаточно ясной.

Преобразуем, далее, выбранное подмножество в схему. Схема будет содержать по одной операции умножения на каждое ребро G_{min}. Ясно, что множители, отвечающие каждому ребру (частное мономов в соседних узлах), входят во множество корневых мономов. Также туда входят первые степени всех переменных, за исключением тех, которые оказались изолированными в G_{min}, как, например, моном $(0, 1, 0)$ на рис. 5. Пример получающейся схемы изображен на рис. 6

После того, как подмножество наименьшего веса выбрано, нам остается добавить операции, соответствующие умножению на коэффициенты и суммированию, а также построить схему для вычисления корневых мономов. Очевидно, в данном случае разделение схемы на две части выглядит несколько нелогично. Мы обсудим этот момент и его влияние на эффективность схемы в разд. 2.6.

Рассмотрим пример. Пусть задан полином $x^2yzw + xy^2zw + xyz^2w + xyzw^2$. ММПД в этом случае построит схему вида $x(xyzw + y^2zw + yz^2w + yzw^2)$, тогда как интуитивно оптимальной будет $xyzw(x + y + z + w)$. Проблема заключается в неспособности данного метода добавлять промежуточные мономы. Если добавить такую возможность, то мы получим интересный метод, предположительно уступающий по эффективности методу аддитивных цепочек, но превосходящий и метод Горнера, и ММПД. В рамках данной

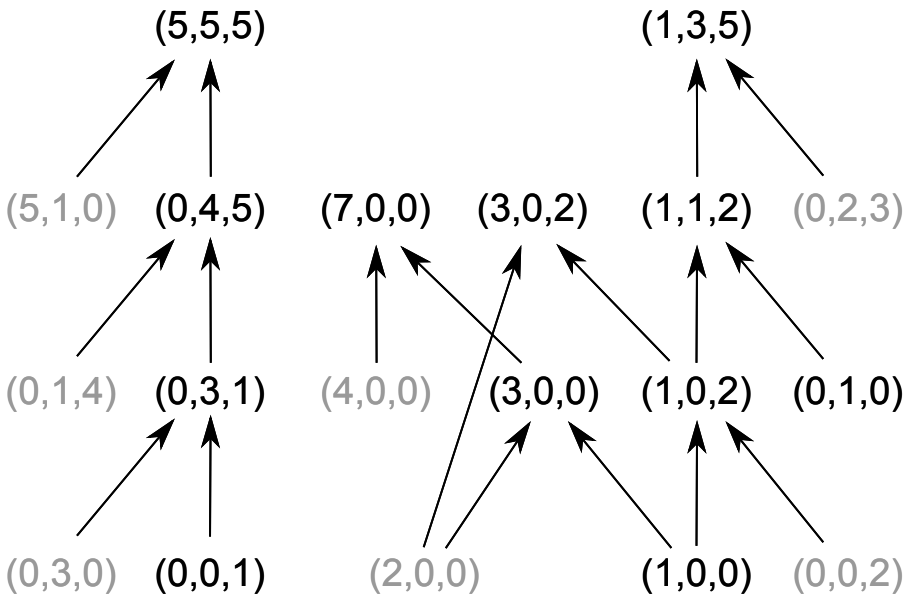

Рис. 6: Схема, построенная по графу на рис. 5. Серым выделены добавочные корневые мономы.

работы мы не будем рассматривать его подробнее.

2.6 CCBM и аддитивные цепочки

Схемы совместного вычисления мономов, как было указано выше, является частным случаем схем вычисления полиномов. В отличие от последних, они содержат лишь операции умножения мономов. Исторически, однако, задача совместного вычисления мономов рассматривалась независимо.

Задача совместного вычисления мономов часто формулируется в терминах *аддитивных цепочек*.

Аддитивной цепочкой (addition chain) в n-мерном пространстве называется последовательность a_i целочисленных векторов, первыми n членами которой являются элементы канонического базиса, а каждый из следующих элементов представляет собой сумму каких-либо двух предыдущих:

$$a_{ij} = \delta_{ij}, \quad i = 1..n, \; j = 1..n$$

$$a_i = a_j + a_k, \quad j \leq k < i, \quad i > n$$

Целочисленные векторы здесь можно рассматривать как мультииндексы мономов. Тогда сложение векторов будет соответствовать умножению мономов.

Лучше всего на сегодняшний день изучены одномерные аддитивные цепочки – они применяются в ряде алгоритмов, а том числе в криптографии, для эффективного возведения в степень (*потенцирования*). Бинарный алгоритм возведения в степень является частным случаем алгоритма на основе аддитивных цепочек. Вопрос о длине $l(m)$ кратчайшей аддитивной цепочки, содержащей число m, достаточно подробно описан в [2]. Для

$l(m)$ известна асимптотическая оценка [15]: $l(m) \sim log(m)$ (здесь и далее логарифмы по основанию 2). Заметим, что бинарный алгоритм дает цепочку длины $2log(m)$ в худшем случае.

Задача о кратчайшей одномерной аддитивной цепочке также часто обобщается следующим образом – вместо одного числа, предлагается найти кратчайшую цепочку, содержащую множество чисел [2]. В англоязычной литературе эта обобщенная постановка называется addition *sequence* problem, и мы будем использовать термин *аддитивная последовательность*, так как более подходящего термина нам неизвестно. Длину такой цепочки обозначают $l(m_1, m_2, ..., m_k)$. Длину кратчайшей k-мерной аддитивной цепочки обозначают $l([m_1, m_2, ..., m_k])$. Интересный результат получен в [21] – показана «эквивалентность» двух последних задач. Именно, для любого k и $\{m_i\}_{i=1}^{k}$, $l([m_1, m_2, ..., m_k]) = l(m_1, m_2, ..., m_k) + k - 1$.

Также установлено [17], что

$$\forall k \, L(\bar{m}_1, \bar{m}_2, ..., \bar{m}_k) \sim log\left(max_{i=1..k}(\bar{m}_i)\right),$$

где $L(\bar{m}_1, \bar{m}_2, ..., \bar{m}_k) = max_{\{m_i < \bar{m}_i\}} l(\{m_i\})$.

В некоторых алгоритмах, таких как DSA, используются многомерные аддитивные цепочки. Также в ряде задач используются аддитивные цепочки с вычитанием – они могут быть короче обычных аддитивных цепочек. Однако, это оправдано лишь в тех случаях, где такая операция допускается целевым доменом и имеет в нем разумную вычислительную сложность.

Также подробно изучен вопрос о сложности построения кратчайшей аддитивной цепочки – эта задача является NP-полной [28]. Существуют и достаточно эффективные приближенные алгоритмы [29, 30, 31].

Схему совместного вычисления мономов, очевидно, можно рассматривать как многомерную аддитивную последовательность, однако построение оптимальной схемы сводится к вычислению кратчайшей аддитивной последовательности лишь в случае постоянного критерия качества, например, при вычислении значений мономов в действительных числах с плавающей точкой фиксированной длины или в конечном поле. Если вычисления производятся в целых числах, кратчайшая аддитивная последовательность уже не соответствует, строго говоря, оптимальной схеме вычисления. В таких задачах, как возведение полинома в степень, неоптимальность таких схем была также подтверждена экспериментально (этот факт упоминается в [2]). Логично предположить, что для задачи редукции полинома, где сложность варьируется еще сильнее, схема, соответствующая кратчайшей аддитивной последовательности, будет далека от оптимальной. Тем не менее, такая схема содержит ценную информацию о структуре рассматриваемого набора мономов, и определенно заслуживает рассмотрения, возможно, с некоторыми модификациями.

2.7 Метод аддитивных цепочек

...

2.8 Схемы специального вида

Для некоторых полиномов специального вида можно построить более эффективные схемы, используя информацию об их структуре. В качестве примера, рассмотрим задачу совместного вычисления мономов для носителя некоторого плотного полинома.

Определение 4. *Полином p называется плотным, если*

$$(\exists x \in V\, x \in s(p)) \wedge (\forall \alpha \in s(p)\, \exists \beta \in s(p), x \in V\, \alpha = x\beta)\,.$$

Значение плотного полинома с n членами может быть вычислено за не более чем $n-1$ операций умножения и $n-1$ операций сложения. Построение соответствующей схемы также может быть реализовано весьма эффективно. Легко видеть, однако, что такую же схему можно построить и методом минимального покрывающего дерева. Мы не будем рассматривать данный алгоритм подробнее в рамках данной работы, так как класс приложений, для которых преимущество в скорости построения схемы вычисления для плотных полиномов дает существенный выигрыш в производительности, достаточно узок.

3 Метод инкрементальной редукции

Редукция зачастую осуществляется относительно набора полиномов, не являющегося базисом Грёбнера. В этом случае результат не единственен и может зависеть от используемого алгоритма редукции. Однако, можно говорить о нормальной форме полинома в более общем смысле, если зафиксировать метод редукции. В данном разделе мы будем использовать термин «нормальная форма» в этом нестрогом смысле, имея ввиду некоторый полином, полученный редукцией данного относительно заданного набора и нередуцируемый относительно него

Также следует обратить внимание на тот факт, что мы используем два различных метода редукции, первый из которых может быть выбран произвольно, а второй – (МИР) – строится на основе первого.

3.1 Задача редукции

Задача редукции полинома p относительно набора полиномов H суть задача поиска остатка от деления этого полинома на них. Простейший подход [1] к его нахождению заключается в последовательном вычитании $h \in H$ из p с некоторыми множителями. Чтобы гарантировать, что редукция в какой-то момент завершится, обычно фиксируют некоторое *допустимое мономиальное упорядочение* и используют на каждом шаге один из полиномов, чей старший член (в смысле указанного упорядочения) делит старший член рассматриваемого полинома, а в качестве множителя используется их частное.

Когда же вычитание выполнить нельзя, старший член полинома перемещается в остаток, и процедура повторяется до тех пор, пока p не окажется равным нулю.

На практике, низкая эффективность алгоритма редукции часто связана с ростом количества промежуточных мономов. В процессе редукции количество членов полинома может интенсивно расти. Рассмотрим, в качестве примера, набор

$$G = \{x_1^2 - (1 + \sum_{i=2}^{m} x_i)\} \cup \{x_i | i = 2..m\}.$$

Теперь редуцируем x^{2n} относительно G. Если применять первое правило, пока это возможно, мы получим полином $(1 + \sum_{i=2}^{m} x_i)^n$, содержащий C_n^{n+m-2} членов, который затем будет редуцирован к 0. Данный пример является искусственным и на практике число промежуточных членов обычно не растет настолько интенсивно, однако существенное влияние этого фактора на эффективность алгоритма редукции установлено.

МИР направлен на сокращение количества промежуточных членов, и в большинстве случаев он позволяет этого достичь, однако можно привести примеры, когда использование обычной редукции более эффективно.

3.2 Описание метода

МИР основан на двух свойствах нормальных форм (1, 2). Первое из них – линейность оператора нормализации. Второе заключается в том, что нормальная форма произведения двух полиномов равна нормальной форме произведения их нормальных форм.

$$N(\lambda p + \mu q) = \lambda N(p) + \mu N(q) \tag{1}$$

$$N(p \cdot q) = N(N(p) \cdot N(q)) \tag{2}$$

Используя эти правила, можно определить пространство нормальных форм по заданному базису. Легко видеть, что оно изоморфно факторалгебре по идеалу, заданному рассматриваемым базисом.

$$\lambda p +_n \mu q = \lambda p + \mu q \tag{3}$$

$$p \cdot_n q = N(p \cdot q) \tag{4}$$

Итак, МИР представляет собой операцию подстановки – или *замены переменных* – в факторалгебре. Данную подстановку можно рассматривать как тождественную – как правило, не изменяются даже имена переменных, – но не следует забывать, что она выполняется в другом пространстве.

Заметим также, что в определении операции умножения в пространстве нормальных форм присутствует оператор нормализации. Ему соответствует некоторый алгоритм редукции, не совпадающий с МИР, так как в противном случае определение последнего было бы некорректным. Мы обсудим этот оператор подробнее в разделе 4.4.

Чтобы редуцировать полином при помощи данного метода, необходимо

- вычислить нормальные формы мономов нижнего уровня,
- выполнить операции согласно схемы, используя (3, 4),
- вернуть результат последней операции в качестве ответа.

3.2.1 Сложность операций умножения

Очевидно, что эффективность метода существенно зависит от используемой схемы, причем аппроксимировать критерий качества в данном случае весьма проблематично – сложность редукции двух мономов, сигнатуры которых отличаются на единицу, может различаться в произвольное число раз.

В двумерном случае множество мономов можно изобразить графически (рис. 7). Множество редуцируемых мономов, которые можно получить из нередуцируемых мономов путем умножения их на первую степень одной из переменных, будем называть *достаточным множеством редуцируемых мономов* (ДМ), так как зная нормальные формы все

мономов в нем, мы можем вычислить нормальную форму любого полинома относительно данного базиса, не выполняя более никаких редукций. ДМ можно рассматривать как своего рода таблицу умножения.

Заметим, что множество нередуцируемых мономов (и, следовательно, ДМ) конечны лишь для нуль-мерных идеалов, и даже если они конечны, вычислять все нормальные формы заранее нет необходимости – мы можем редуцировать те мономы, которые появляются в процессе вычислений. Это реализуется посредством механизма кеширования нормальных форм.

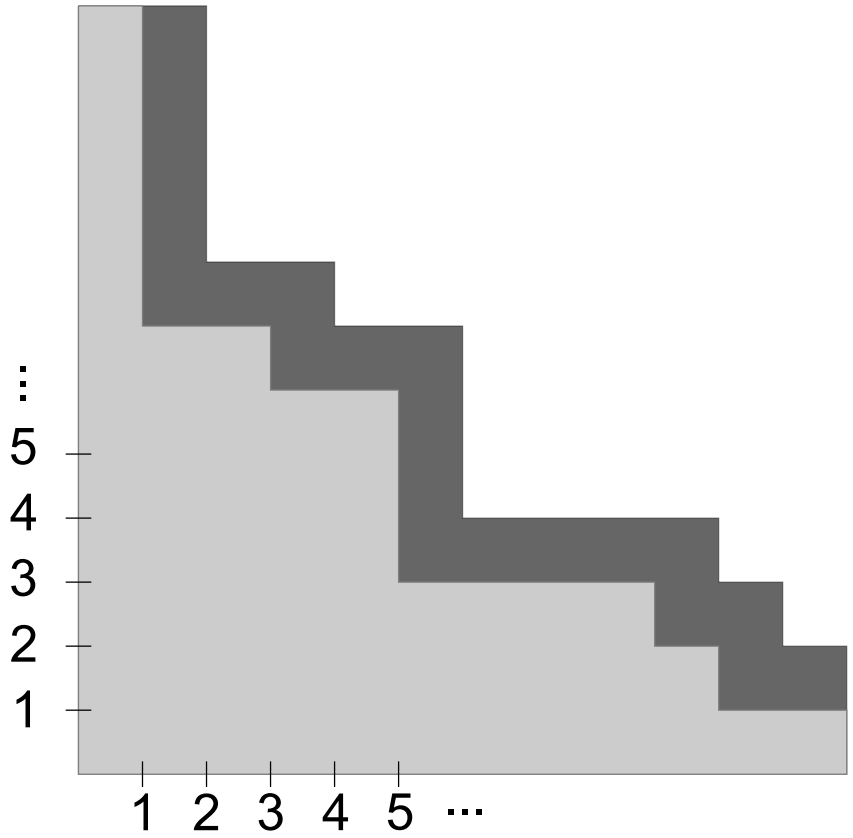

Рис. 7: Светло-серым изображена область нередуцируемых мономов, темно-серым – ДМ.

В данной работе мы реализуем такую схему кеширования (см. 4.4) и оцениваем ее эффективность. Следует заметить, что данная идея не нова – она используется, в частности, в [5]. Есть также определенные основания полагать, что некоторая схожая схема кеширования реализована в системе компьютерной алгебры Singular.

Что касается операций сложения и умножения на число, они имеют, в большинстве случаев, пренебрежимо малую стоимость по сравнению с операцией умножения.

3.3 Наращивание схемы

МИР демонстрирует высокую эффективность в наших экспериментах. Однако, с тем, чтобы узнать его сильные и слабые стороны, следует рассматривать фактическую сложность выполнения отдельных операций. Мы полагаем, это позволит как улучшить алгоритм вы-

числения в соответствии с заданной схемой, так и построить более точный приближенный критерий сложности, с тем чтобы генерировать более эффективные схемы.

В разделе **??** мы обсудим некоторые эксперименты с данным алгоритмом. Наблюдая работу МИР, мы пришли к выводу, что, вероятно, модификация схемы в процессе выполнения редукции может дать существенный прирост производительности. Дело в том, что при редукции могут возникать новые мономы, которых не было в исходном полиноме, и их редукция может потребовать значительного времени. В то же время, велика вероятность, что мы уже знаем нормальную форму одного или нескольких мономов, которые делят рассматриваемый моном. Используя эти мономы, мы можем «достроить» фрагмент схемы, с тем чтобы включить в нее новый моном.

Развивая эту идею, мы неизбежно приходим к заключению, что строить схему лучше прямо в процессе ее выполнения, так как мы обладаем в этот момент дополнительной информацией, которая не была доступна ранее. Например, дополнительные мономы, добавленные в схему в процессе редукции, могут позволить нам достичь целевых мономов более коротким путем.

Схему можно наращивать двумя способами – «в ширину» или «в глубину». Какой из подходов предпочтителен – неясно, – это требует экспериментальной проверки. Кроме того, не вполне ясно, достаточно ли планировать схему на каждом шаге для всех оставшихся мономов, какого-то одного из них, или некоего подмножества, а также, насколько критично качество промежуточной схемы в этом случае.

К сожалению, мы не имеем возможности вполне исследовать данный вопрос в рамках данной работы.

4 Детали реализации

В рамках данной работы была реализована библиотека для работы с полиномами над полем рациональных чисел или конечным полем с учетом специфики рассматриваемой задачи на языке `C++`. При помощи данной библиотеки был реализован ряд описанных ранее алгоритмов с целью сравнения их эффективности между собой и с системой компьютерной алгебры Maple 14 [26], а также выявления потенциальных путей повышения эффективности рассматриваемых алгоритмов.

При реализации библиотеки использованы некоторые базовые контейнеры, механизм ввода-вывода и средства анализа эффективности кода из библиотеки Indigo API [27], распространяемой под лицензией GPL.

4.1 Прототипирование

На стадии прототипирования описанные методы построения схем – метод Горнера и ММПД, – а также алгоритм вычисления значений полинома по заданной схеме были реализованы на языке Python. Позднее они были также перенесены на C++ с целью повышения эффективности, а также во избежание дублирования отдельных процедур работы с полиномами.

4.2 Представление мономов

Представление мономов в программных пакетах компьютерной алгебры изучено достаточно хорошо. Анализ эффективности различных представлений можно найти в [32]. Мы используем представление в виде вектора степеней с полной степенью.

Анализа состава набора мономов, используемых в процессе редукции полиномов, показал, что многие мономы низких степеней используются повторно. Чтобы снизить потребление памяти и повысить эффективность библиотеки в целом, мы храним все уникальные мономы, используемые в данный момент времени в специальной структуре данных на основе хэш-таблицы с подсчетом ссылок. Таким образом, мономам присваиваются уникальные номера, что существенно упрощает реализацию механизма кеширования нормальных форм.

На основе анализа работы алгоритма редукции для полиномов высоких степеней мы также пришли к выводу, что удаление мономов при обнулении счетчика ссылок не является необходимым, так как количество используемых мономов обычно растет практически в течение всего времени работы алгоритма, после чего резко падает. Мы не выполняем формального сравнения такого подхода с традиционным, так как используемые в данной работе алгоритмы в любом случае требуют наличия у мономов уникальных ключей, т.е. накладных расходов на поддержание структуры с уникальными мономами не избежать. Однако, мы допускаем, что такой подход может быть плохо приспособлен для систем

компьютерной алгебры общего назначения.

Поддерживаются лексикографическое и градуированное обратное лексикографическое упорядочения.

4.3 Представление полиномов

Полиномы представлены в виде списков с общим пулом элементов. Это означает, что при создании нового полинома выделяется лишь небольшой объем памяти на стеке, а память, выделенная в общем пуле для членов полинома, может быть использована повторно после уничтожения данного объекта – это особенно полезно при реализации алгоритма редукции, где возникает большое количество промежуточных полиномов.

В качестве коэффициентов могут выступать элементы конечного поля, чей модуль не превышает 2^{31}, либо рациональные числа произвольной точности. Последние реализованы посредством библиотеки GMP [24], которая представляет собой де-факто стандарт вычислений произвольной точности и используется в таких системах компьютерной алгебры как Mathematica, Maple и Singular, а также во множестве других программных пакетов и приложений.

4.4 Кеширование нормальных форм мономов

Как было сказано выше, используемый способ представления мономов автоматически дает нам уникальные идентификаторы для них, поэтому реализация кеша нормальных форм мономов не представляет сложности. Внимания заслуживает вспомогательный алгоритм редукции, используемый в нашей реализации МИР. Нашей целью было гарантировать, что нормальная форма любого нужного нам монома будет вычислена лишь однажды (если только мы не удалим ее из кеша явно, например, с целью высвобождения памяти).

Итак, вспомогательный алгоритм редукции получает на вход моном m и должен вернуть в качестве результата его нормальную форму, также записав ее в кеш. Нормальные формы всех промежуточных мономов, потребовавшихся нам в процессе редукции, также должны попасть в кеш.

Соответствие предложенного алгоритма (2) указанным условиям легко проверить. Ясно, что нормальная форма всякого монома, вычисленная данным алгоритмом, будет помещена в кеш. Остается удостовериться, что редукция одного и того же монома не будет выполнена дважды. Это возможно лишь в случае, если данный моном окажется промежуточным продуктом редукции самого себя. Однако все мономы p строго меньше m в смысле заданного упорядочения, и, по индукции, все мономы, могущие встретиться в процессе редукции мономов из p также строго меньше m. Следовательно, указанная ситуация невозможна и алгоритм удовлетворяет указанным требованиям.

Заметим, что практическая реализация данного алгоритма не является рекурсивной, так как для полиномов высоких степеней это могло бы привести к переполнению стека в

Algorithm 2: AuxReduce(m,G,T)

Input: моном m, базис G, кеш нормальных форм T

Output: нормальная форма монома m

if *HasKey(T,m)* **then**
  return $T[m]$ `// monomial cached`
end
foreach $g \in G$ **do**
  $lm \leftarrow LeadingMonomial(g)$
  if lm *divides* m **then**
    $p \leftarrow m - m \cdot g/lm$
    $q \leftarrow 0$
    foreach *term* $t \in p$ **do**
      $q \leftarrow q + Coefficient(t) \cdot AuxReduce(Monomial(t), G, T)$ `// recursive call`
    end
    $T[m] \leftarrow q$
    return $T[m]$
  end
end
$T[m] \leftarrow m$ `// monomial is irreducible`
return $T[m]$

процессе работы алгоритма.

4.5 Реализация СВП

Представление СВП в программе практически в точности повторяет формальное определение – это последовательность операций вида:

- вычислить нормальные формы мономов $m_1, m_2, ..., m_k$ и записать в первые k ячеек памяти
- вычислить сумму или произведение полиномов в ячейках i и j и записать в ячейку r
- вычислить произведение полинома в ячейке i и коэффициента f и записать в ячейку r

Классы, реализующие вычисление значений полинома, МИР и подсчет количества операций в схеме, представляют собой интерпретаторы такой программы.

4.6 Поддерживаемые операции для мономов и полиномов

Приведем список операций, реализованных для мономов:

- создание монома с заданной мультистепенью
- создание монома на основе строки, например $x1^4 * x2^7$
- вычисление произведения, частного и наибольшего общего делителя мономов
- проверка делимости и равенства
- сравнение в заданном упорядочении
- получение степени конкретной переменной и полной степени
- вывод монома в текстовой форме.

Полиномы поддерживают

- добавление и удаление членов
- загрузку из строки и сохранение в строку
- сложение и умножение на число (коэффициент)
- перебор членов
- получение старшего члена
- сортировку согласно заданного упорядочения.

Список литературы

[1] Кокс Д., Литтл Дж., О'Ши Д. Идеалы, многообразия и алгоритмы. Москва, Мир 2000

[2] Knuth D. The Art of Computer Programming, vol. 2, Addison-Wesley 1981

[3] Knuth D. Evaluation of Polynomials by Computer, Communications of the ACM, vol. 5, issue 12, ACM 1962, pp. 595-599

[4] Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов, Москва, Мир, 1979

[5] Клискунов А.Н. Эффективные вычисления нормальных форм в алгоритме Бухбергера. Магистерская диссертация, СПбГПУ ФизМех, 2006

[6] Vasiliev N.N., Joint Evaluations of Sparse Polynomials and the Synthesis of Evaluating Programms. Proceedings of International Conference Systems and Methods of Analytical Computations with Computers in Theor. Phys., Dubna, 1985, pp. 154-159, (In Russian)

[7] Vasiliev N.N., Creation of efficient symbolic-numeric interface. Lecture Notes in Computer Science. V378 , Springer, 1989, pp. 118-119

[8] Vasiliev N.N., The System for the Synthesis of evaluating programms for polynomial-trigonometric expressions. Proceedings of International Conference Analytical Computations with Computers in Theor. Phys., Dubna, 1983. pp.120-124

[9] Bini D., Pan V. Parallel polynomial computations by recursive processes, Proc. ACM SIGSAM Intern. Symp. on Symbolic and Algebraic Comp., 294, 1990

[10] Bini D., Gemignani L., Pan V. Improved parallel computations with matrices and polynomials, Proc. 18-th Intern. Symposium on Automata, Languages and Programming, Lectures Notes in Computer Science, 510, 520-531, Springer 1991

[11] Dimitrov V.S., Jullien G.A., Miller W.C. Algorithms for multiexponentiation based on complex arithmetic, in Proceedings of the 13th Symposium on Computer Arithmetic, IEEE 1997

[12] Moeller B. Algorithms for Multiexponentiation, Technical report, TU Darmstadt 2001

[13] Bernstein D. Pippenger's Exponentiation Algorithm, `http://cr.yp.to/papers.html#pippenger`, unpublished

[14] Vasiliev N.N. Complexity of monomial evaluations and duality. in «Computer Algebra in Scientifig Computing», V.G.Ganzha (ed.) 479-485, Springer, 1999

[15] Brauer, O.M. Addition-Subtraction Chains, Diplomarbeit, University Paderborn 2001

[16] Кочергин В.В. Об асимптотике сложности аддитивных вычислений систем целочисленных линейных форм, Дискретный анализ и исследование операций, Апрель-Июнь 2006. Серия 1. Том 13, № 2. 38–58

[17] Кочергин В.В. О вычислении наборов степеней, Дискретная математика, том 6, выпуск 2, 1994, с. 129-137

[18] Белага Э.Г. О вычислении значений многочленов от одного переменного с предварительной обработкой коэффициентов. Проблемы кибернетики, вып. 5. М., Физматлит, 1961. С. 7-16

[19] Пан В.Я. Некоторые схемы для вычисления многочленов с действительными коэффициентами. Проблемы кибернетики, вып. 5. М., Физматлит, 1961. С. 17-30

[20] Александров Д.Е., Галкин В.В., Зобнин А.И., Левин М.В. Распараллеливание матричных алгоритмов вычисления базисов Грёбнера. Фундаментальная и прикладная математика, 2008, том 14, № 4, с. 35—64

[21] Olivos J. On vectorial addition chains. Journal of Algorithms, Elsevier, 1981, 13-21

[22] Axiom: the scientific computation system, `http://www.axiom-developer.org/`

[23] Singular, computer algebra system, `http://www.singular.uni-kl.de/`

[24] The GNU multiple precision arithmetic library, `http://gmplib.org/`

[25] Sage, open-source mathematics software systems, `http://www.sagemath.org/`

[26] Maple, computer algebra system, `http://www.maplesoft.com/`

[27] Indigo API, `http://opensource.scitouch.net/indigo/`

[28] Downey P., Leong B., Sethi R. (1981). Computing sequences with addition chains. SIAM Journal on Computing 10 (3): 638–646.

[29] Bos J., Coster M. Addition chain heuristics, in Proceedings of CRYPTO'89, Springer 1995

[30] Gordon D. A survey of fast exponentiation methods, Journal of Algorithms 27, Elsevier 1998, pp. 129-146

[31] Cruz-Cortes N., Rodriges-Henriques F., Jurez Morales R., Coello C. Finding optimal addition chains using a genetic algorithm approach, 2005, unpublished, `http://delta.cs.cinvestav.mx/~francisco/cis2005_nareli.pdf`

[32] Bachmann O., Schoenemann H. Monomial representations for Groebner basis computation, in Proceedings of International Symposium on Symbolic and Algebraic Computation, ACM 1998