Задачи дискретной оптимизации с квазиблочными матрицами

Д.В. Лемтюжникова[1], Д.В. Ковков[2]

Аннотация—Рассмотрены алгоритмы для решения целочисленных квазиблочных задач оптимизации. Проанализированы современные методы декомпозиции. Исследован метод Финкельштейна и её модификации для выделения квазиблочных структур. Проанализирована эффективность локального элиминационного алгоритма для задач большой размерности. Рассмотрены особенности применения параметрической оптимизации. Исследовано влияние порядка решения подзадач на работу алгоритма. Проведены тестовые эксперименты решения задач целочисленного линейного программирования большой размерности для точных, приближенных и эвристических алгоритмов. Приведены эксперименты для распараллеливания локального элиминационного алгоритма с помощью ГРИД–технологий. Показаны примеры задач, которые не могут быть решены без применения технологии распараллеливания.

Ключевые слова: декомпозиция, целочисленное программирование, квазиблочная структура, локальный элиминационный алгоритм, порядок элиминации, ГРИД

I. Введение

Рассматриваются оптимизационные задачи с матрицами квазиблочной структуры с целочисленными переменными. Они являются результатом применения критерия для локального элиминационного алгоритма (ЛЭА) О.А. Щербины [1] и имеют блочно–лестничную или блочно–древовидную структуру. Сами эти структуры появляются после соответствующих процедур перестановки столбцов [2] для задач с разреженными матрицами.

Особенностью ЛЭА является использование локальной информации об окрестностях так называемых связывающих переменных, то есть переменных, принадлежащих одновременно нескольким окрестностям. Поэтому ЛЭА позволяет получать глобальное решение задачи с помощью локальных вычислений, которые обычно являются решениями соответствующих подзадач. ЛЭА в общем виде представляется как последовательное исключение переменных с сохранением информации о них. Процедура ЛЭА состоит из первой и второй части. В первой части переменные исключаются согласно заданному порядку, при этом вычисляемая информация о каждой переменной сохраняется в виде локальных решений (чаще всего таблиц), а в конце процесса вся полученная информация представляется в виде значения критерия. Во второй части находится глобальное решение исходной задачи по найденным в первой части таблицам с локальными решениями, которые обеспечивают достижение критерия, представленного в первой части. В данной статье ЛЭА будет рассматриваться в основном класс бинарных задач ДО, а именно — булевых задач линейного программирования:

$extr\{cx \mid Ax \gtreqless b, x_j \in \{0,1\}, j \in N\}$.

Здесь c — вектор целевой функции, A — матрица ограничений, b — вектор ограничений.

В данной работе представлены результаты численного тестирования, которые позволяют выявить специфику применения рассматриваемого подхода.

II. Выделение квазиблочной структуры

С улучшением алгоритмов за счёт параллельных вычислений растёт использование декомпозиций и их модификаций для решения задач смешанного целочисленного программирования. Наиболее известные среди них: декомпозиция Бендерса [3], [4], [5], двойственная декомпозиция [6], ячеевидная декомпозиция [7] и декомпозиция Фенхеля [8] и кросс-декомпозиция [9], [10], лагранжева декомпозиция [11]. Также используются комбинации различных методов, например объединение алгоритмов прогрессивного хеджирования и двойственной декомпозиции [12] или объединение декомпозиции Бендерса и метода отсечений Гомори [13]. Также следует отметить работы по параллельной реализации декомпозиционных методов дискретной оптимизации [14], [15].

В [2] исследуется метод Финкельштейна, а также предлагается её модификация для выделения блочно–лестничной структуры в разреженных задачах. Основная идея данного вида декомпозиции заключается в изучении возрастающих окрестностей переменных и их последующей компоновке в блоки. Первая модификация связана с устранением основных недостатков декомпозиции Финкельштейна. Она заключается в использовании предельного значения для числа связывающих переменных между блоками, чтобы размеры сепараторов не были слишком большими, а также в исключении пустых блоков. Вычислительный эксперимент показал улучшение структуры тестовых задач для первой модификации декомпозиции Финкельштейна относительно декомпозиции без модификации. Существенно уменьшается число блоков и размеры сепараторов.

В данной статье предлагается вторая модификация Финкельштейна для выделения блочно–древовидной

*Работа выполнена при поддержке РФФИ, № 16-51-53093

[1]Д.В. Лемтюжникова — старший преподаватель в МАИ. darabbt@gmail.com

[2]Д.В. Ковков — научный сотрудник в ВЦ им. А.А. Дородницына ФИЦ «Информатика и управление» РАН. dmkovkov@gmail.com

структуры. Полученная с помощью алгоритма Финкельштейна блочно–лестничная структура имеет вид цепи. Чтобы разделить блоки на подблоки таким образом, чтобы образовалась блочно–древовидная структура, воспользуемся алгоритмом выделения подблоков. На первом шаге исследуется сепаратор текущего блока. Затем, если существуют несвязанные ограничения в блоке–потомке, сепаратор делится на несколько сепараторов в зависимости от несвязанных ограничений, после чего и сам блок-потомок делится на подблоки согласно полученным сепараторам.

III. Эффективность локального элиминационного алгоритма

Вопрос эффективности ЛЭА подробно рассматривается в работе О.А. Щербины [1]. В данной статье рассматривается эксперимент, проведённый для тестовых задач ЦЛП с целью исследования эффективности ЛЭА с встроенным решателем SYMPHONY [1] для решения подзадач относительно самого решателя SYMPHONY, который решает всю задачу целиком без ЛЭА.

В ходе эксперимента все тестовые задачи ЦЛП с бинарными переменными имели блочно–древовидную структуру, сгенерированную искусственным образом. Все блоки в отдельно взятой задаче имели одинаковое количество переменных, а также сепараторы одинакового размера внутри каждой задачи. Размер сепараторов для одних и тех же параметров задачи изменялось для получения оценки времени решения задачи при увеличении размера сепаратора. Тестовые задачи ЦЛП генерировались исходя из заданного общего количества переменных, связывающих переменных между блоками, а также числа ограничений. Размеры и число блоков вычислялись согласно числу переменных и ограничений исходной задачи. С помощью генератора случайных чисел задавались остальные компоненты задачи: коэффициенты целевой функции, коэффициенты матрицы ограничений и правых частей для каждого блока. Каждая тестовая задача решалась с использованием следующих алгоритмов. Первый алгоритм — это базовый решатель SYMPHONY для задач частично—целочисленного линейного программирования (ЧЦЛП). Второй алгоритм — ЛЭА, который использовал с решатель SYMPHONY для решения подзадач, соответствующих блокам. Третий алгоритм — вариация второго алгоритма, где решатель SYMPHONY использовался с поддержкой технологии тёплого старта (ТС).

В результате было установлено явное преимущество второго и третьего алгоритмов над первым. В частности, эксперимент показал, что второй алгоритм становился менее эффективным из–за увеличения объема перебора при решении подзадач в блоках, если увеличивать количество связывающих переменных в задачах с одинаковым числом переменных и размером блоков. В этом случае разумно использовать третий алгоритм, а именно ЛЭА в сочетании с решателем SYMPHONY, когда используются принципы параметрической оптимизации (технологию ТС). Дело в том, что соответствующие одному и тому же блоку задачи ЦЛП отличаются одна от другой только правыми частями для разных значений связывающих переменных. Алгоритм получает информацию о решении и использует её для анализа последующих задач, что позволяет решать каждую задачу не полностью, а частично при переборе значений связывающих переменных. Значит с помощью параметрического программирования можно существенно увеличить производительность ЛЭА. Однако результат эксперимента оказался неоднозначным. Время решения большинства задач с использованием второго и третьего алгоритмов практически сопоставимо. Для некоторых тестовых задач параметрическое программирование оказалась неэффективным. Было замечено также, что что при малых размерностях эффективность ЛЭА отсутствует.

[1] https://projects.coin-or.org/SYMPHONY

IV. Порядок решения подзадач

В теории локальных алгоритмов Журавлёва выделены две основные теоремы — теорема единственности и теорема мажорантности [16]. В теореме единственности утверждается, что результат вычисления основных предикатов локального алгоритма с монотонными функциями (которые не возрастают/ не убывают на заданном множестве) не зависит от порядка рассмотрения элементов множества. То есть, каком бы порядке мы не решали подзадачи в задаче, оптимальное решение будет получено. В теореме мажорантности доказывается существование для всякого класса локально равных алгоритмов с одинаковой памятью наилучшего (мажорантного) локального алгоритма, т.е. алгоритма, который по заданной фиксированной системе окрестностей вычисляет заданные основные предикаты при фиксированных вспомогательных предикатах всегда, когда это делает любой другой алгоритм из рассматриваемого класса. То есть от того, в каком порядке мы решаем подзадачи зависит скорость нахождения оптимального решения. О.А. Щербина показал, что доказательство этой теоремы имеет неконструктивный характер [1], т.е. не позволяет осуществить прямое построение эффективного наилучшего алгоритма. В его диссертационной работе сформулирован критерий существования оптимального порядка решения подзадач, который заключается в следующем. Оптимальный порядок решения подзадач существует тогда и только тогда, когда процедуре решения соответствует дерево, каждая вершина которого соответствует подзадаче. Подмножеству переменных каждой подзадачи должна соответствовать клика в графе ограничений исходной задачи. Граф ограничений строится так: множеству вершин соответствуют переменные или группы переменных с одинаковыми окрестностями, которые соединяются ребром в случае, если они одновременно находятся хотя бы в одном ограничении исходной задачи. Клика — это подмножество графа ограничений, в котором любые две вершины соединены ребром.

Дерево, которое позволяет определить оптимальный порядок решения подзадач, можно выделить с помощью алгоритма поиска клик в графе. Для этого

существует целый класс ПО, в частности, с открытым кодом. Например, NetworkX даёт приближённое решение этой задачи (процедура max_clique), в maxClique заложены точные алгоритмы, а с помощью OpenOpt можно получить точные и приближённые решения, при этом можно управлять процессом: есть возможность указать рёбра, которые должны быть включены. Но сама задача о клике является NP–трудной задачей, поэтому для больших задач ДО необходим другой способ определения наилучшего порядка решения подзадач.

Одним из способов декомпозиции задачи ДО является так называемая «элиминационная игра». Этот термин впервые введён Партером [17] в 1961 году как интерпретация метода исключения Гаусса на графах. Граф, полученный из графа ограничений переменных с помощью удаления некоторой вершины x_k и всех ребер, исходящих из нее, а затем соединения ребрами всех ранее не соседних вершин в окрестности x_k, называется x_k — элиминационным графом G^k. Описанная операция называется элиминацией вершины x_k. Последовательность всех элиминированных вершин называется порядок элиминации α. В данном контексте элиминационная игра — это процесс преобразования графа ограничений с помощью ЛЭА. В терминах «элиминационной игры» ЛЭА может быть кратко записан следующим образом. На первом шаге необходимо перенумеровать переменные согласно порядку элиминации. Далее на каждом шаге изменять граф взаимосвязей путем добавления ребер для превращения окрестности исключаемой вершины в клику.

Определить порядок элиминации можно с помощью эвристик. В данной статье предлагается вычислительный эксперимент, в котором сравнивается скорость решения задач с помощью ЛЭА, использующего различные порядки элиминаций, полученные с помощью эвристик: алгоритм упорядочивания минимальной степени [18], алгоритм рекурсивного разбиения [19], эвристика минимального пополнения [20], лексикографический поиск в ширину [21] и поиск по максимальной степени [22]. Тестовые задачи ДО генерировались на основе уже существующих гиперграфов из библиотеки CSP [2]. Указанная библиотека содержит различные классы гиперграфов для задач удовлетворения ограничений, среди которых были реальные задачи для приложений в промышленности (DaimlerChrysler, NASA, ISCAS) и искусственно созданные примеры задач. В данном эксперименте всего было взято 150 примеров случайно выбранных задач. Время решения одних и тех же задач с разными эвристиками сравнивалось и выбиралась лучшая из них. Таким образом, для алгоритма рекурсивного разбиения минимальное время работы алгоритма не было достигнуто, для алгоритма упорядочивания минимальной степени – 7 раз, для лексикографического поиска в ширину – 11 раз, для поиска по максимальной степени – 39 раз и для эвристики минимального пополнения – 93 раза.

[2]http://www.dbai.tuwien.ac.at/proj/hypertree

V. Приближённый и эвристический подходы

Одним из недостатков локального блочно–элиминационного алгоритма является полный перебор по множествам связывающих переменных, что обусловливает возможность большого объема перебора при большом количестве связывающих переменных. Преодолеть этот недостаток ЛЭА и способствовать успешному решению практически важных задач ДО специальной структуры с большим числом связывающих переменных могло бы введение процедуры, позволяющей «предсказать» искомые оптимальные (или близкие к оптимальным) значения связывающих переменных. Зная значения связывающих переменных, задачи ДО, соответствующие блокам (окрестностям), можно было бы решать отдельно, независимо друг от друга. Впервые эта идея была введена О.А. Щербиной [1]. Понятно, что возможность узнать каким–то образом именно оптимальные значения связывающих переменных представляется сомнительной; тем не менее можно найти значения связывающих переменных, близкие к оптимальным, с помощью замены задач ДО, соответствующих блокам, на их релаксации [23] и последующего решения построенной таким образом задачи ДО со специальной структурой с помощью ЛЭА. Найденные для релаксированных блоков оптимальные значения связывающих переменных (все возможные наборы перебираются полностью, однако трудоемкость выполнения каждого шага существенно снижена за счет решения не исходных, а релаксированных — более легко решаемых подзадач). В качестве релаксаций используется линейная релаксация, когда условия $x_j = \{0; 1\}$ заменяются неравенствами $0 \leq x \leq 1$. Получается задача частично–целочисленного программирования, которая решается итеративно. Непрерывное решение затем округляется.

Целью вычислительного эксперимента является провести оценку эффективности использования эвристического локального элиминационного алгоритма (ЭЛЭА) и жадного алгоритма, а также сравнить вышеперечисленные приближенные алгоритмы с точным алгоритмом. Все задачи для вычислительного эксперимента являются задачами ЦЛП с булевыми переменными, которые имеют блочно–лестничную структуру. Блоки в каждой задаче имеют одинаковое количество переменных и ограничений, а также одинаковое число переменных в сепараторах между блоками. Длина сепаратора меняется для одних и тех же параметров задачи, поскольку позволяет оценить эффективность каждого из приближенных алгоритмов в зависимости от числа связывающих переменных.

Тестовые задачи генерировались по заданному числу переменных, числу ограничений и размеру сепараторов между блоками. Исходя из количества переменных и ограничений вычислялись размеры блоков и их количество. Далее с помощью датчика случайных чисел генерировались коэффициенты целевой функции, коэффициенты матрицы ограничений и правых

частей для каждого из блоков. Библиотека LES (Local Elimination Solver) версии v0.1.1 [3] использовалась для вычисления задач бинарного целочисленного линейного программирования (BILP problem). Матрица ограничений исходной задачи изображена на рисунке 1. Здесь k — количество блоков; i_p — количество строк

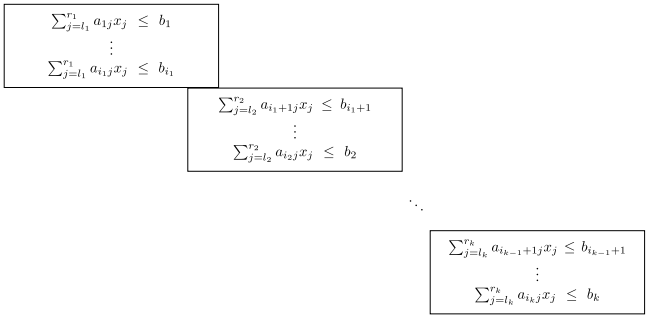

Рис. 1: Матрица исходной задачи ДО

в p–том блоке, $p = 1, \ldots, k$; i и j — номера строки и столбца соответственно, $i = 1, \ldots, n$, $j = 1, \ldots, m$; l_p и r_p — левая и правая граница блоков, $l_p < l_{p+1} \leq r_p < r_{p+1}$; причем $i_k = n$, $l_1 = 1$ и $r_k = m$. Далее исследуем работу каждого из алгоритмов.

Рассмотрим принцип работы ЭЛЭА. После того, как построен декомпозиционный граф данной задачи, на его основе строится новый граф, в вершинах которого находятся релаксированные подзадачи. В каждой полученной подзадаче содержится только одно ограничение, определяемое вектором, каждый элемент которого — это сумма элементов первоначальной матрицы ограничений по строкам. Матрица ограничений релаксационной задачи представлена на рисунке 2. Значения общих переменных, полученные

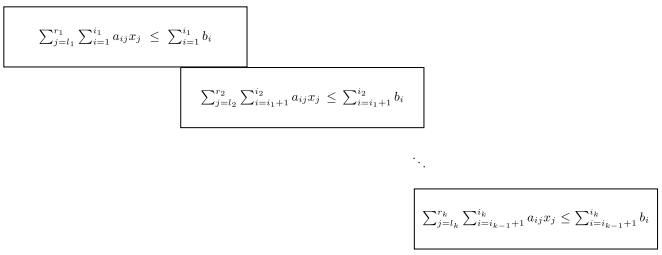

Рис. 2: Матрица релаксированной задачи

в результате решения задачи с модифицированным деревом, являются компонентами вектора решений исходной задачи. Затем исходная задача упрощается следующим образом. Значения общих переменных подставляются в задачу так, что изменяются правые части ограничений, а из целевой функции исключаются общие переменные. То есть, в результате данного преобразования получается k не зависящих друг от друга задач, как показано на рисунке 3. Здесь x_j^* — значения переменных, полученных после решения релаксированной задачи. Таким образом, вместо исходной задачи решаются релаксированные подзадачи, что позволило существенно сокращает время работы алгоритма.

[3]https://github.com/robionica/les

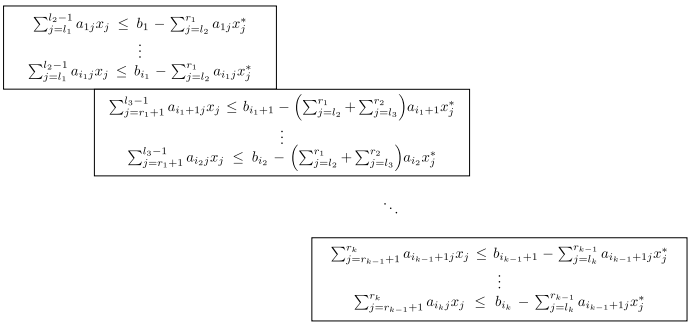

Рис. 3: Матрица упрощенной задачи ДО

Таблица I: Результаты сравнения алгоритмов ЭЛЭА и жадного алгоритма

					Точность	
	n	m	k	s	ЭЛЭА	Жадный
1	100	300	10	6	99,31	98,67
2	100	300	10	10	98,40	97,71
3	100	300	10	14	99,02	97,28
4	100	600	10	6	99,85	98,99
5	100	600	10	10	99,51	98,16
6	100	600	10	14	99,59	97,54
7	200	500	10	6	99,49	98,26
8	200	500	10	10	99,34	98,61
9	200	500	10	14	98,72	98,174

Жадный алгоритм позволяет упростить нахождение значений общих переменных. Для этого после декомпозиции исходной задачи для каждой переменной задаётся вес w следующим образом. Пусть x_p — некоторая переменная исходной задачи, D_p — множество индексов ограничений, таких что коэффициент при x_p ненулевой, k_p — число блоков, в которые входит x_p, $p = 1, 2, \ldots, m$. Тогда весом переменной будем называть следующую величину: $w = (c_p \cdot \sum_{i \in D_p} b_i)/(k_p \cdot \sum_{i \in D_p} a_{ip})$.

Отсортируем все веса в порядке убывания, затем будем поочередно задавать переменным значение 1, пока решение удовлетворяет условию задачи. После этого исходная задача упрощается и решается k задач, как и в ЭЛЭА.

Результаты вычислительного эксперимента представлены в таблице I. Для проверки эффективности приближенных алгоритмов в качестве тестовых задач были взяты пакеты задач, задачи в которых отличаются только размерами сепараторов между блоками. Это позволяет узнать, каким образом увеличение размера сепараторов влияет на точность приближенных алгоритмов.

В таблице I находятся результаты вышеописанного эксперимента. m — число ограничений, n — число переменных задачи, k — число блоков соответствующей структуры, s — максимальный сепаратор, точность — точность приближенных алгоритмов относительно точного ЛЭА.

Рассмотрим тестирование ЛЭА с препроцессингом (ЛЭАпр). ЛЭАпр уменьшает размерность исходной задачи путем исключения переменных и ограничений. Вначале проводится декомпозиция задачи согласно рисунку 1. Обозначим блоки задачи — Ω_t, где $t = 1, 2, \ldots, k$ — количество блоков. Зададим вес для Ω_t: определим вес блока как сумму коэффициентов

целевой функции при переменных x_j, которые входят в данный блок.

$$w_t = \sum_{j:x_j \in \Omega_t} c_j$$

Разобьем исходную задачу на $3k-2$ следующим образом. Первый и последний блоки разбиваются на две подзадачи, а все остальные — на три: одна из них полностью состоит из локальных переменных данной подзадачи, а две другие — из общих переменных. Правые части ограничений разбиваются в соответствии с весами каждого блока. Рассмотрим произвольный внутренний блок Ω_t, $t = 2, \ldots, k-1$. Пусть L_t — множество локальных переменных, S_{t_l} и S_{t_r} — множества общих переменных таких, что $S_{t_l} \in \Omega_{t-1} \bigcap \Omega_t$ и $S_{t_r} \in \Omega_t \bigcap \Omega_{t+1}$. Тогда после разбиения на подзадачи блок Ω_t будет выглядеть следующим образом (рис. 4):

$$\sum_{j:j\in S_{t_l}} a_{i_b j} x_j \le b_{i_b 0}$$
$$\sum_{j:j\in S_{t_l}} a_{i_b j} x_j \le b_{i_b 0}$$

$$\sum_{j:j\in L_t} a_{i_b j} x_j \le b_{i_b 1}$$
$$\sum_{j:j\in L_t} a_{i_b j} x_j \le b_{i_b 1}$$

$$\sum_{j:j\in S_{t_r}} a_{i_b j} x_j \le b_{i_b 2}$$
$$\sum_{j:j\in S_{t_r}} a_{i_b j} x_j \le b_{i_b 2}$$

Рис. 4: Разбиение блока на подзадачи

Где $i_b = i_{t-1} + 1$ и $i_e = i_t$, $b_{ij} = \frac{b_i}{\sum_{j=0,1,2} b_{ij}}$. Объединим все подзадачи, которые имеют одинаковые множества переменных. После решения этих подзадач исключим из первоначальной задачи все переменные, которые приняли значение 1. Затем соответствующим образом изменим правые части ограничений. Полученное значение целевой функции является искомым.

Так как упрощенная задача может иметь избыточные ограничения, а также переменные, не удовлетворяющие ограничениям, необходимо использовать правила предварительного анализа исходной задачи — препроцессинг. Препроцессинг состоит из трех этапов: исключение переменных, не удовлетворяющих хотя бы одному из ограничений, исключение избыточных ограничений (при этом в вектор решений добавляются переменные, которые в любом случае будут принимать значение единицы) и исключение переменных, которые не входят в новые ограничения задачи. Затем решается упрощенная задача с помощью решателя SCIP. Полученные значения переменных записываются в вектор решений, а решение упрощенной задачи добавляется к значению целевой функции.

В таблице II находятся результаты вышеописанного эксперимента. m — число ограничений, n — число переменных задачи, k — число блоков соответствующей структуры, s — максимальный сепаратор, точность — точность приближенного алгоритма относительно точного ЛЭА, ускорение — во сколько раз приближенный алгоритм работает быстрее, чем ЛЭА.

Анализ полученных результатов показал, что при достаточно больших сепараторах между блоками квазиблочной задачи ЦЛП приближенные ЛЭА в сочетании с современными решателями позволяют решать задачи быстрее, чем используемые точные алгоритмы сами по себе при решении всей задачи. Кроме того, с увеличением размерности задач ускорение жадного и эвристического алгоритма становится более сопоставимым. При этом эвристический алгоритм намного чаще «угадывает» значения сепараторов, нежели

Таблица II: Результаты, полученные с помощью решателя SCIP и ЛЭА с препроцессингом

№	n	m	k	s	Точность	Ускорение
1	40	160	10	3	100,00	0,41
2	40	160	10	5	98,03	0,85
3	40	160	10	7	96,61	1,84
4	80	200	10	3	99,56	0,62
5	80	200	10	5	98,92	0,94
6	80	200	10	7	96,82	3,41
7	120	300	15	3	99,36	1,22
8	120	300	15	5	98,24	1,61
9	120	300	15	7	98,12	6,46
10	100	300	10	3	99,75	2,02
11	100	300	10	5	98,69	2,08
12	100	300	10	7	98,06	1,81
13	150	625	25	3	99,77	0,38
14	150	625	25	5	98,86	4,37
15	150	625	25	7	98,41	76,28
16	100	600	10	3	99,96	1,68
17	100	600	10	5	99,11	3,12
18	100	600	10	7	98,87	7,11
19	200	500	10	3	99,87	4,97
20	200	500	10	5	99,91	37,76
21	200	500	10	7	99,12	157,83

жадный алгоритм, что позволяет точно решить соответствующую подзадачу. Значит можно предположить, что для задач с многократно большими размерностями более эффективным для использования представляется эвристический локальный элиминационный алгоритм.

VI. Распараллеливание блочных задач

Наличие блоков в рассматриваемых постановках с блочно–древовидной и блочно–лестничной структурой и концепция ЛЭА о независимом решении оптимизационных задач для отдельных блоков приводит к естественному решению осуществлять параллельные решения на различных процессорах одновременно. Для этой цели используется ГРИД–система с гибким распределением свободных вычислительных ресурсов [24]. Указанная система работает по следующему принципу. Имеется центральный компьютер. С помощью некоторой операционной системы через Интернет к нему подключаются ресурсы других компьютеров, число которых достигает полутора тысяч. В фиксированный момент среди тех порядка нескольких сотен представляют свои вычислительные ресурсы. Это решает важнейшую проблему незагруженности компьютеров в данный момент времени (грубая оценка даёт 90% ресурсов).

Для параллельной реализации ЛЭА использовалась облачная платформа Everest, поддерживающая публикацию, выполнение и композицию вычислительных приложений в распределенной среде [25], [26]. Одной из отличительных черт платформы является возможность запуска приложений на произвольных комбинациях внешних вычислительных ресурсов, подключенных пользователями. Программирование ЛЭА для платформы Everest осуществлялось с помощью языка алгебраического программирования AMPL [4]. Так как AMPL не решает задачи непосредственно, в качестве соответствующего внешнего «решателя» использовался пакет SCIP — широко используемая

[4] www.ampl.com

система с открытым кодом [5]. При этом транслятор AMPL использовался не только для формулировки основной задачи и всех вспомогательных подзадач, но и для управления сценарием расчетов в распределенной среде солверов, подключенных к Everest, с помощью подсистемы AMPLX [6], доступной в исходных кодах.

В результате проведенного вычислительного эксперимента было отмечено, что для задач с блочно–древовидной и блочно–лестничной структурой ЛЭА с распараллеливанием, использующий решатель SCIP гораздо эффективнее, чем решатель SCIP без ЛЭА. Задачи с блочно–лестничной структурой содержали 100 000 переменных и 100 ограничений и решались в среднем за 6 часов. Задачи разбивались на 5 подзадач с глубиной дерева — 5. При этом подзадачи могли содержать максимально 20 000 переменных 20 ограничений, размер сепаратора и число вершин потомков — 4. Задачи с БД структурой содержали 50 000 переменных и 100 ограничений и решались в среднем за 17 минут. Задачи разбивались на 15 подзадач с глубиной дерева — 5. При этом подзадачи могли содержать максимально 10 000 переменных 20 ограничений, размер сепаратора — 4, а число вершин потомков — 12.

При анализе решения задач была установлена следующая проблема: большое число подзадач обрабатывается очень быстро. Причина в том, что большое число подзадач несовместны. К сожалению, препроцессинг AMPLa не позволяет регулярным образом «пропустить» заведомо несовместную подзадачу. Поэтому препроцессинг приходится отключать и «поручать» проверку несовместности решателям. Если выполнять проверку на совместность условий заранее, число вычислений существенно сократится и решение исходной задачи будет получено быстрее.

Основным результатом данного эксперимента является реализация ЛЭАП для решения задач ДО со специальной структурой с помощью GRID. В результате проведенного вычислительного эксперимента было отмечено, что для задач с блочно–древовидной и блочно–лестничной структурой ЛЭАП гораздо эффективнее, чем решатель без ЛЭАП.

VII. Заключение

В этой статье рассмотрены задачи с квазиблочными матрицами. Также исследована эффективность локального элиминационного алгоритма (ЛЭА) и влияние порядка исключения переменных на его скорость. Рассматриваются различные модификации ЛЭА: параллельная, приближённая и эвристическая. Поставлен ряд новых задач в рассматриваемой области, в частности, оценка количества вычислений при распараллеливании, выделение классов задач с полиномиальной сложностью при выборе порядка исключений в локальном алгоритме, применение теорем о составе квазиблочной структуры для оптимального выбора количества процессоров при распараллеливании, а также оценка количества вычислений для алгоритмов выделения блочно–лестничной и блочно–древовидной структуры.

[5]http://scip.zib.de/

[6]https://gitlab.com/ssmir/amplx

Список литературы

[1] Щербина О. А. Локальные элиминационные алгоритмы для разреженных задач дискретной оптимизации: дис. ... доктора физ.–мат. наук: 05.13.17. — М., 2011. — 361 p.

[2] Лемтюжникова Д. В., Свириденко А. В., Щербина О. А. Алгоритм выделения блочно–древовидной структуры в разреженных задачах дискретной оптимизации // Таврический вестник информатики и математики. — 2012. — Vol. 1. — P. 44–55.

[3] Ciré A., Coban E., Hooker J.N. Mixed integer programming vs. logic–based benders decomposition for planning and scheduling // CPAIOR 2013: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. — 2013. — P. 325–331.

[4] Chu Y., You F. Integration of production scheduling and dynamic optimization for multi–product cstrs: Generalized benders decomposition coupled with global mixed–integer fractional programming // Computers & Chemical Engineering. — 2013. — Vol. 58. — P. 315–333.

[5] Network–constrained ac unit commitment under uncertainty: A benders' decomposition approach / Amin Nasri, S. Jalal Kazempour, Antonio J. Conejo, Mehrdad Ghandhari // IEEE Transactions on Power Systems. — 2016. — Vol. 31, no. 1. — P. 412–422.

[6] On parallelizing dual decomposition in stochastic integer programming / M. Lubin, Kipp Martin, Cosmin G. Petra, Burhaneddin Sandikci // Operations Research Letters. — 2013. — Vol. 41, no. 3. — P. 252–258.

[7] Park J., Karumanchi S., Iagnemma K. Homotopy–based divide–and–conquer strategy for optimal trajectory planning via mixed-integer programming // IEEE Transactions on Robotics. — 2015. — Vol. 31, no. 5. — P. 1101–1115.

[8] Ntaimo L. Fenchel decomposition for stochastic mixed–integer programming // Journal of Global Optimization. — January, 2013. — Vol. 55, no. 1. — P. 141–163.

[9] Mitra S., Garcia-Herreros P., Grossmann I. E. A novel cross–decomposition multi–cut scheme for two–stage stochastic programming // Computer Aided Chemical Engineering. — 2014. — Vol. 32. — P. 241–246.

[10] Mitra Sumit, Garcia-Herreros Pablo, Grossmann Ignacio E. A cross–decomposition scheme with integrated primal–dual multi–cuts for two–stage stochastic programming investment planning problems // Mathematical Programming. — 2016. — Vol. 157, no. 1. — P. 95–119.

[11] A lagrangian decomposition approach for the pump scheduling problem in water networks / Bissan Ghaddar, Joe Naoum-Sawaya, Akihiro Kishimoto et al. // European Journal of Operational Research. — 2015. — Vol. 241, no. 2. — P. 490–501.

[12] Integration of progressive hedging and dual decomposition in stochastic integer programs / G. Guo, Gabriel Hackebeil, Sarah M. Ryan et al. // Operations Research Letters. — 2015. — Vol. 43, no. 3. — P. 311–316.

[13] Zhang Minjiao, Küçükyavuz Simge. Finitely convergent decomposition algorithms for two-stage stochastic pure integer programs // SIAM J. Optim. — 2014. — Vol. 24, no. 4. — P. 1933–1951.

[14] Evtushenko Yuri, Posypkin Mikhail, Sigal Israel. A framework for parallel large-scale global optimization // Computer Science-Research and Development. — 2009. — Vol. 23, no. 3-4. — P. 211–215.

[15] Posypkin M., Khrapov N. Branch and bound method on desktop grid systems // Young Researchers in Electrical and Electronic Engineering (EIConRus). — 2017. — P. 526–528. — URL: http://neo.lcc.uma.es/workshops/PPSN2012/papers/p2.pdf.

[16] Журавлев Ю. И. Избранные научные труды. — М.: Магистр, 1998. — 420 p.

[17] Parter S. The use of linear graphs in gauss elimination // SIAM Review. — 1961. — Vol. 3. — P. 119–130.

[18] Amestoy P. R., Davis T. A., Duff I. S. An approximate minimum degree ordering algorithm // SIAM Journal on Matrix Analysis and Applications. — 1996. — Vol. 17, no. 4. — P. 886–905.

[19] George J. A. Nested dissection of a regular finite element mesh // SIAM J. Numer. Anal. — 1973. — Vol. 10. — P. 345–367.

[20] Jégou P., Ndiaye S. N., Terrioux C. Computing and exploiting tree–decompositions for (max–)csp // Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP-2005). — 2005. — P. 777–781.
[21] Rose D. J., Tarjan R., Lueker G. Algorithmic aspects of vertex elimination on graphs // SIAM J. on Computing. — 1976. — Vol. 5. — P. 266–283.
[22] Tarjan R. E. Simple linear–time algorithms to test chordality of graphs, test acyclity of hypergraphs, and selectively reduce acyclic hypergraphs // SIAM J. Comput. — 1984. — Vol. 13. — P. 566–579.
[23] Gorry G. A., Shapiro J. F., Wolsey L. A. Relaxation methods for pure and mixed integer programming problems // Management Science. — 1972. — Vol. 18, no. 5. — P. 229–239.
[24] Афанасьев А.П. [и др.]. Концепция многозадачной грид–системы с гибким распределением свободных вычислительных ресурсов суперкомпьютеров // Известия РАН. — 2017. — no. 4. — P. 229–239.
[25] Koutsopoulos I., Papaioannou T. G, Hatzi V. Modeling and optimization of the smart grid ecosystem // Foundations and Trends in Networking. — 2016. — Vol. 10, no. 2–3. — P. 115–316. — URL: http://dx.doi.org/10.1561/1300000042.
[26] Salah A., Hart E. A modified grid diversity operator for discrete optimization and its application to wind farm layout optimization problems // Companion Material Proceedings Genetic and Evolutionary Computation Conference, GECCO 2016 (Denver, CO, USA, July 20–24) / Ed. by Tobias Friedrich, Frank Neumann, Andrew M. Sutton. — 2016. — P. 977–982.

Problems of discrete optimization with quasiblock matrices

D.V. Lemtyuzhnikova, D.V. Kovkov

Abstract—We consider algorithms for solving integer optimization problems with quasi–block structure. Modern decomposition approaches are analyzed. We study Finkelstein decomposition and it variations for discovering quasi–block structure. Efficiency of local elimination algorithm for large–scale problem is analyzed. Specific details of application of parametric optimization are provided. Dependency of order of solving subproblems on the algorithm performance is studied. We provide results of numerical experiments for solving large–scale linear programming problems by exact, approximate, or heuristic algorithms. Also we present experiments for parallelization of local elimination algorithm using grid computing approach. We discuss some problems which cannot be solved efficiently without parallelization techniques.

Keywords—decomposition, integer programming, quasiblock structure, local elimination algorithm, elimination order, GRID