ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

РАЗРАБОТКА АЛГОРИТМА ПОИСКА ПРЕДЕЛЬНЫХ ГРАНИЦ КАРЬЕРОВ РУДНЫХ МЕСТОРОЖДЕНИЙ НА ОСНОВЕ МЕТОДОВ ПЛАВАЮЩЕГО КОНУСА И ЛЕРЧА-ГРОССМАНА

Выпускная квалификационная работа
обучающегося по направлению подготовки 02.03.02 «Фундаментальная информатика и информационные технологии»
очной формы обучения, группы 07001401
Дроника Виталия Игоревича

Научный руководитель:
к.т.н., доцент Михелёв В.М.

БЕЛГОРОД 2018

Оглавление

ВВЕДЕНИЕ

При проектировании добычи полезных ископаемых открытым способом одной из важнейших задач является определение оптимальных границ карьеров рудных месторождений. Решение этой задачи позволяет определить максимальную возможную экономическую выгоду от добычи полезных ископаемых без учёта затрат времени, производственных ограничений, материальных затрат на обслуживание техники и оплату труда и т.д. На основе решения этой задачи в дальнейшем расчитывается целесообразность и экономическая выгода добычи ископаемых с учётом перечисленных факторов, определяется местоположение фабрик переработки, транспортных путей, отвалов и других узлов предприятия.

Наиболее часто используемые в настоящее время методы решения этой задачи обладают рядом недостатков: алгоритмы, обладающие приемлемым временем обработки моделей (модификации плавающего конуса) не всегда дают максимальную оптимальную оболочку карьера, что может привести к неверной оценке экономической выгоды и затрат на добычу; с другой стороны, методы, обеспечивающие качественный результат (алгоритм Лерча-Гроссмана) имеют очень большую вычислительную сложность, что делает их неэффективными при расчёте больших моделей месторождений. Из вышеперечисленного можно сделать вывод, что существует необходимость в разработке новых методов решения данной задачи, которые давали бы оптимальный результат, при этом обеспечивая приемлемое время обработки моделей больших месторождений.

Целью данной работы является разработка и реализация алгоритма поиска предельных границ карьеров рудных месторождений на основе существующих методов, удовлетворяющего поставленным условиям оптимальности решения и времени вычислений.

В рамках работы будет проведено теоретическое обоснование а также исследование вычислительной сложности предлагаемого метода.

Для достижения поставленной цели в ходе выполнения работы необходимо решить ряд задач:

- Изучить теоретические основы задачи поиска предельных границ карьеров рудных месторождений;
- Провести анализ и сравнение существующих методов решения данной задачи;
- Разработать новый алгоритм, используя комбинацию существующих методов поиска;
- Реализовать разработанный алгоритм для проведения вычислительных экспериментов;
- Провести вычислительные эксперименты;
- Сравнить результаты работы приложения с результатами реализаций существующих методов;
- Предложить возможные варианты улучшения предложенного метода.

Данная работа разделена на четыре главы. В первой главе «Анализ предметной области» приводится постановка задачи поиска предельных границ карьеров рудных месторождений как части задачи оптимизации границ карьеров в системе добычи ископаемых открытым способом, рассматриваются наиболее часто используемые в настоящее время методы решения данной задачи с определением их основных достоинств и недостатков.

Во второй главе «Разработка теоретических основ метода поиска предельных границ карьеров» производится математическая постановка задачи оптимизации границ карьеров, приводится описание схем работы

разработанного метода, а также вычисляется ассимптотическая оценка вычислительной сложности алгоритма.

В третьей главе «Программная реализация метода поиска предельных границ карьеров» описываются основные этапы разработки приложения, реализующего разработанный метод, а также обосновывается выбор инструментальных средств разработки.

В четвёртой главе «Проведение вычислительных экспериментов» приводятся результаты исследований эффективности приведённого метода, описание наборов тестовых данных и аппаратных средств, на которыз проводились вычислительные эксперименты. Кроме того, проводится сравнение времени и эффективности работы с реализациями методов плавающего конуса и Лерча-Гроссмана.

Работа состоит из 65 страниц, 20 рисунков, 4 таблиц, одного приложения.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Задача оптимизации границ карьеров в системе недропользования

Одной из важнейших задач, решаемых в процессе проектирования добычи полезных ископаемых открытым способом является определение наиболее выгодной с экономической точки зрения последовательности извлечения породы и расчёт оптимальных границ проектируемого карьера. Для выполнения расчётов по оптимизации на основе результатов горных выработок и опробования скважин с использованием интерполяционных методов построения [35] строится дискретная блочная модель месторождения (рисунок 1.1), в которой блоки соответствуют некоему объёму залегающей породы. В такой модели блоки имеют стоимостную оценку, которая зависит от процентного содержания в нём полезного сырья и породы. На рисунке 1.1 цветом градацией цветов от синего к красному обозначена стоимостная оценка блоков: синим отмечены блоки с наименьшим значением выгоды, красным – блоки с наибольшим содержанием полезных ресурсов.

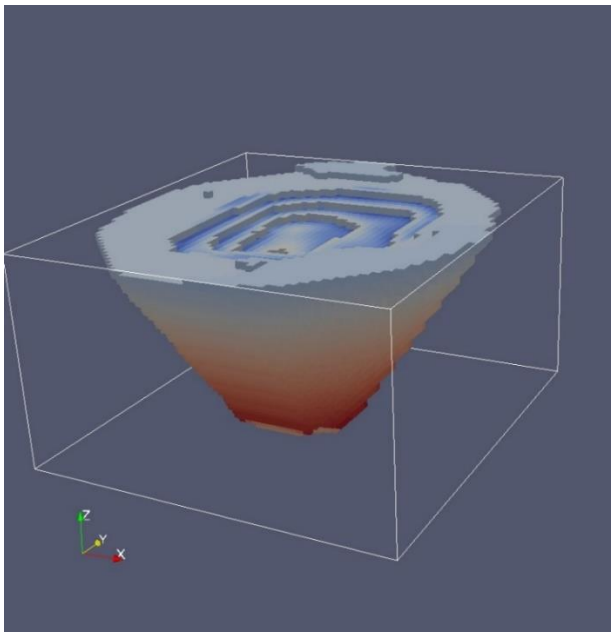

Рис. 1.1. Пример блочной модели месторождения

В решении задачи оптимизации выделяют задачу поиска предельных границ карьера (UPIT), то есть форму карьера в конце цикла добычи ископаемых предприятием. Эта задача состоит в определении наиболее прибыльного множества извлекаемых блоков породы и руды при соблюдении ограничений прецедентности [26]. Решение этой задачи даёт наиболее оптимальную оболочку из блоков разрабатываемого месторождения, при этом не учитываются такие факторы, как временные затраты, производственные ограничения и т.д.

При разработке ископаемых необходимо соблюдение неких условий, гарантирующих, что стенки карьера будут устойчивыми и оборудование будет иметь доступ к зоне раскопки. Прецедентные ограничения между блоками подразумевают, что при добыче залегающего на некой глубине блока, необходимо прежде извлечь множество блоков, лежащих над ним в соответствии с условиями устойчивости стенок и необходимости доступа оборудования. Это ограничение имеет свойство транзитивности, то есть если для извлечения блока A необходимо извлечь множество блоков α, а для извлечения блоков α необходимо извлечь множество блоков β, то для извлечения блока A необходимо извлечь блоки множества β [24].

В соответствии с прецедентными ограничениями существует две схемы извлечения блоков. Первая схема основана на удалении пяти блоков над исходным, при этом угол наклона стенок карьера лежит в пределах от 45° до 55° (рисунок 1.2-а). Вторая схема предполагает удаление девяти блоков над текущим и угол наклона меняется в диапазоне от 35° до 45° (рисунок 1.2-б) [4]. Схема извлечения выбирается в зависимости от технических ограничений при добыче ископаемых, залегающих в определённой породе.

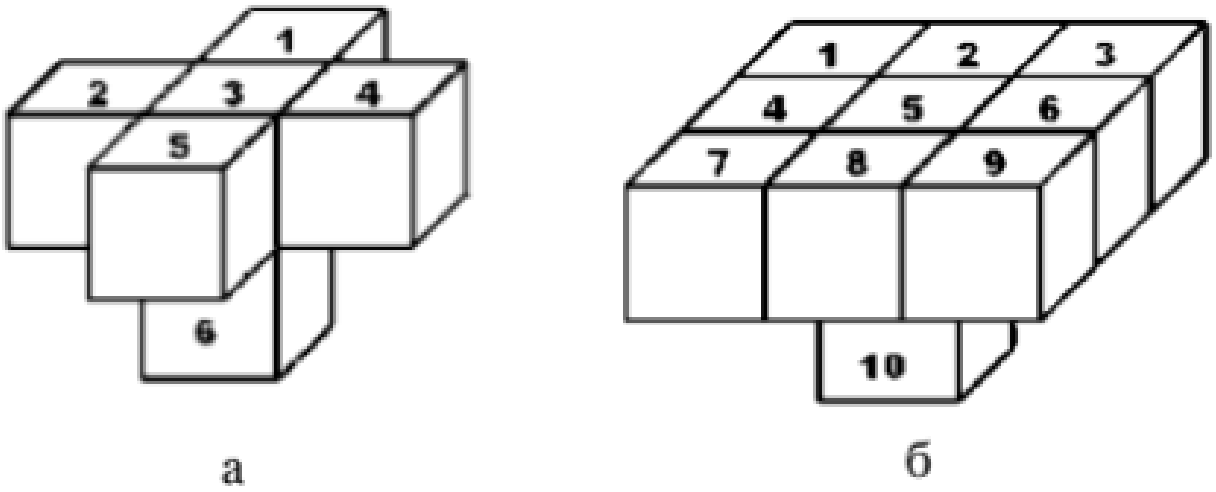

Рис. 1.2. Схемы извлечения блоков, основанные на удалении пяти блоков выше заданного (а) или на удалении девяти блоков выше заданного (б)

1.2 Основные методы нахождения предельных границ карьеров рудных месторождений.

Чаще всего для решения поставленной задачи применяются различные модификации алгоритмов плавающего конуса и Лерча-Гроссмана. Данные алгоритмы имеют определённые достоинства, однако также и ряд недостатков. Все рассматриваемые методы работают с блочными моделями месторождений.

Алгоритм плавающего конуса

Алгоритм плавающего конуса широко применяется в настоящее время в связи с простотой его реализации и небольшой вычислительной сложности.

Метод перебирает блоки в модели послойно, начиная с самого верхнего слоя, заканчивая самым нижним, при этом в одном слое порядок перебора блоков не влияет на результат. При нахождении блока с положительным значением строится усечённый перевёрнутый конус с вершиной в данном

блоке, таким образом выясняется, какие блоки должны быть добыты прежде, чем будет извлечён рассматриваемый блок [37]. Целью построения конуса является определение потенциальной выгоды от добычи этого блока. На рисунке 1.3 представлен пример построения конуса, зелёным отмечен блок с положительным весом – вершина конуса, жёлтым – блоки, входящие в конус, красной линией отмечена граница конуса.

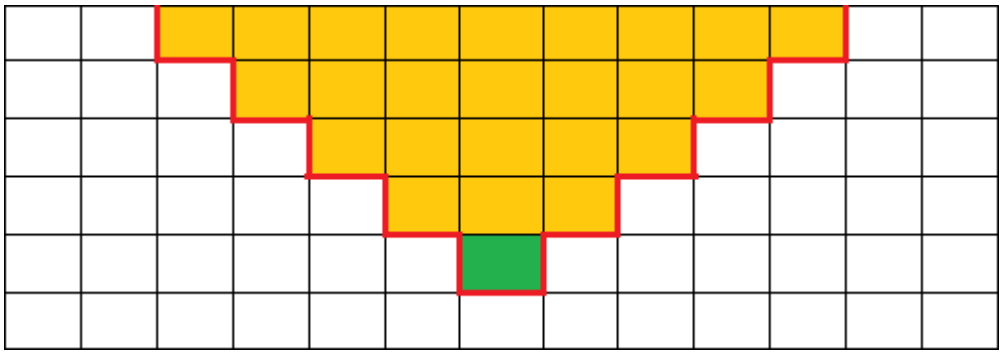

Рис. 1.3. Пример построения конуса

При построении конуса расчитывается его вес как сумма всех блоков, входящих в данный конус, в том числе и вершина. В случае, если вершина находится в самом верхнем слое модели, блок сразу добавляют к решению.

После этого анализируется вес построенного конуса. В случае, если вес положительный - осуществляется проверка существования пересечения данного конуса с другим конусом с отрицательным весом. Если такого пересечения не существует, данный конус добавляется к решению задачи, а блоки, входящие в состав этого конуса исключаются из дальнейших расчётов. Данные действия повторяются до тех пор, пока не будут рассмотрены все возможные пересечения, либо вес конуса не станет равен отрицательному значению. Если после выполнения этой процедуры вес конуса не стал отрицательным, данный конус добавляется к решению задачи. В случае, если вес конуса отрицательный либо равен нулю – алгоритм переходит к рассмотрению оставшихся положительных блоков модели.

Работа алгоритма завершается в том случае, если были рассмотрены все положительные блоки в блочной модели. В результате будет получено множество блоков, принадлежащих к конусам, имеющим положительные вес. Данное множество будет являтся решением задачи оптимизации, то есть оптимальным карьером. Сумма всех блоков оптимального карьера даст максимальную возможную прибыль от разработки месторождения.

Данный метод очень прост для реализации и понимания. Тем не менее, он обладает рядом недостатков. Например, при увеличении размера модели, а также количества положительных блоков, существенно увеличивается время нахождения решения. Однако самым серьёзным недостатком является то, что при использовании его для месторождений с беспорядочным распределением полезных ископаемых метод зачастую не способен найти максимальное оптимальное решение. В связи с этим, алгоритм не применяется в первоначальной формулировке, а используются различные модификации [12].

Алгоритм Лерча-Гроссмана

В настоящее время при решении задачи оптимизации карьеров большую популярность имеет алгоритм Лерча-Гроссмана [6], основанный на теории графов. В данном алгоритме блочная модель месторождения представляется в виде взвешенного ориентированного графа G =(V,A). Каждый блок модели i представляется в виде вершины графа с весом b_i, который определяет чистую выгоду при добыче данного блока. Две вершины i и j соединяются направленной дугой с началом в i и концом в j, если блок i не может быть извлечён раньше, чем блок j. Пример месторождения и соответствующего ему месторождения представлен на рисунке 1.4. Зелёным цветом на блочной модели выделены положительные блоки, а на графе – соответствующие им сильные вершины.

а)

-5	2	-5	-5	-5	-5	-5	-5
-5	-5	-5	-5	-5	8	-5	-5
-5	-5	-5	4	-5	-5	-5	-5
-5	11	-5	-5	-5	-5	-5	2

б)

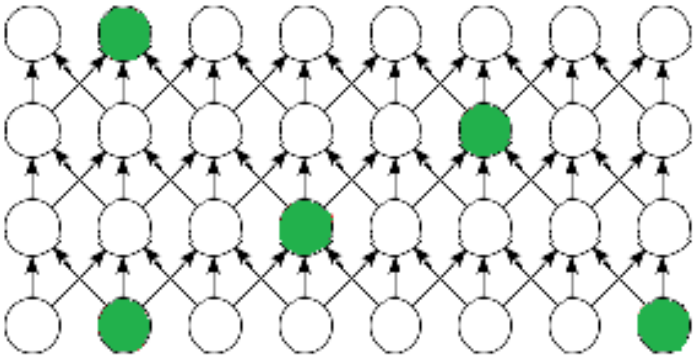

Рис. 1.4. Пример блочной модели месторождения (а) и соответствующего ей графа (б)

Важной концепцией теории графов, используемой в данном алгоритме является замыкание, которое определяется как как некое множество вершин C, таких, что если $u \in C$ и (u,v) есть дуга в графе, то $v \in C$. Вес замыкания равен сумме весов вершин в замыкании. В контексте горных работ замыкание представляет собой некий контур карьера, суммарный вес которого равен сумме весов блоков, входящих в замыкание [17]. Таким образом задача оптимизации карьеров сводится к нахождению максимального замыкания взвешенного ориентированного графа, характеризующего месторождение.

Одним из важных понятий теории, положенной в основу алгоритма, является понятие фиктивного корня. Данный узел нужен для того, чтобы быть корневым узлом дерева на этапе обработки. Необходимо учитывать, что блока, соответствующего данной вершине, на самом деле, не существует. Предполагается, что он должен быть ниже основания модели и иметь отрицательную стоимость.

Для работы алгоритма вводятся понятия «сильных» и «слабых», положительных и отрицательных дуг графа. Положительной дугой называют дугу, которая, выходя из корневого блока, направлена в сторону самого верхнего слоя. Понятия направления в пространстве не входят в теорию графов, а вводятся лишь в терминах графов алгоритма.

Сильной дугой можно назвать положительную дугу, если она поддерживает положительный вес, либо отрицательную дугу, поддерживающую отрицательный вес. Все остальные дуги считаются слабыми

Сильной вершиной является вершина, которая связана с корневым фиктивным блоком по меньшей мере одной сильной дугой. Сильные верширы, связанные между собой образуют группы. Такие группы называются ветвями. Изначально каждый блок представляет собой ветвь, а сильными блоками являются только те блоки, которые имеют положительный вес. Кроме того, используется понятие нормализованного дерева. Дерево считается нормализованным в том случае, если все сильные рёбра берут начало в блоке фиктивного корня.

В процессе работы алгоритм исследует рёбра, ведущие из сильных блоков к слабым. При добавлении таких рёбер к существующим ветвям нарушается условие того, что текущий граф является нормализованным. Для того, чтобы устранить данное явление в структуру графа вносятся изменения связей между блоками. После того, как не осталось ни одного ребра, ведущего от сильной к слабой вершине, множество сильных блоков считается оптимальным карьером.

Алгоритм Лерча-Гроссмана широко используется благодаря тому, что способен в любом случае найти максимальную оптимальную оболочку карьера. Однако данный метод имеет большую вычислительную сложность и расчёт моделей месторождений большого размера занимает очень много времени [17]. Кроме того, данный алгоритм хоть и прост для восприятия, тем не менее сложен для программирования.

Метод максимизации псевдопотока

В основе данного метода лежит та же теория графов, что и в алгоритме Лерча-Гроссмана. Нормализованые деревья алгоритма Лерча-Гроссмана были преобразованы к более общей модели сетевого потока, в основе которого лежит концепция псевдопотока [5]. Сетевой псевдопоток удовлетворяет ограничениям пропускной способности, однако в нём нарушаются условия баланса потока и создаются дефициты или избытки в узлах.

Алгоритм состоит из двух шагов – шага поглощения и шага нормализации. Эти шаги повторяются до тех пор, пока вершинами сильных ветвей не будет сформировано максимальное замыкание графа G. Массы, поддерживаемые корневым узлом сильного дерева, считаются неким псевдопотоком и выталкиваются к слабому корню, а затем к фиктивному корневому узлу. На каждом этапе с каждой вершиной и ребром деревя связано несколько переменных, представляющих вес исходящего из ребра или вершины поддерева, тип связывающего вершину с его предком ребра (положительное или отрицательное), а также тип вершины или ребра (сильная либо слабая).

Данный алгоритм является дальнейшим развитием алгоритма Лерча-Гроссмана [5], направленным на увеличение производительности при сохранении качества результата. К недостаткам алгоритма можно отнести высокую сложность понимания и реализации, а также использование сложных древовидных структур представления модели. Существуют различные варианты реализации алгоритма максимального псевдопотока, наиболее известными являются методы максимизации псевдопотока с верхней и нижней меткой.

1.3 Постановка задачи

Проблема поиска предельных границ карьеров является актуальной в настоящее время, поскольку имеющиеся методы решения данной задачи имеют ряд недостатков, что приводит либо к неточностям при проектировании, либо к необходимости использовать менее точные модели месторождений. В связи с этим, имеется необходимость в разработке нового метода, который обеспечивал бы корректный результат для любых типов месторождений и имел бы меньшую вычислительную сложность, чем существующие алгоритмы. Предположительно, для достижения качественного результата можно использовать алгоритм Лерча-Гроссмана либо алгоритм максимизации псевдопотока, а ускорение вычислений обеспечить уменьшением размера обрабатываемой структуры, вследствие чего необходимо изменить смысловое значение узлов и дуг графа.

Таким образом, в рамках данной работы следует разработать новый подход к решению задачи поиска предельных границ карьеров рудных месторождений на основе существующих методов, реализовать его в виде приложения для ЭВМ, провести вычислительные эксперименты и сравнить результаты с другими подходами, а также предложить возможные способы улучшения предложенного метода

ГЛАВА 2. РАЗРАБОТКА ТЕОРЕТИЧЕСКИХ ОСНОВ МЕТОДА ПОИСКА ПРЕДЕЛЬНЫХ ГРАНИЦ КАРЬЕРОВ

2.1 Математическая постановка задачи

Для аналитического описания задачи оптимизации необходимо ввести функции плотности v,c,p, определяемые в каждой точке пространства месторождения:

v(x, y, z) – стоимость руды, располагающейся в данной точке месторождения;

c(x, y, z) – стоимость извлечения породы;

m(x, y, z) = v(x, y, z) – c(x, y, z) – чистая прибыль от добычи данного объёма руды и породы.

Пусть V – семейство объёмов, соответствующих семействам поверхностей S. Среди всех возможных семейств объёмов необходимо найти то, которое приводит к максимальному значению интервал:

$$\int_V m(x, y, z)dxdydz \qquad (1)$$

Это соответствует выражению, что необходимо найти границы карьера рудного месторождения, при соблюдении которых предприятие по добыче получит максимальную прибыль.

При решении задачи с применением ЭВМ используют дискретную блочную модель рудного месторождения, получаемую с помощью полигональных, триангуляционных и интерполяционных методов построения моделей на основании результатов предварительно проведённых опробовываний скважин и горных выработок. Каждому блоку такой модели соответствует вес, отражающий чистую прибыль от его добычи с учётом

содержания в нём полезного материала, стоимости его извлечения, переработки и прибыли от продажи добытого сырья.

На рисунке 2.1 приведён пример одного из вертикальных слоёв блочной модели месторождения.

-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	2	4	1	1	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	3	8	11	4	0	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	6	18	10	3	-1	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	5	19	22	10	4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	9	8	8	5	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	1	6	4	1	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4

Рис. 2.1. Пример фрагмента блочной модели месторождения

На рисунке зелёным отмечены блоки, имеющие положительные значения веса. Содержания полезного материала в таких блоках достаточно, чтобы получать прибыль от их добычи. Блоки серого цвета имеют отрицательный вес. Это означает, что добывая такие блоки, предприятие затрачивает матеральных ресурсов больше, чем получает прибыли от тих добычи. Блоки с нулевым значением также считаются отрицательными, так как хоть материальные затраты на их извлечение и равны прибыли, получаемой при продаже, тем не менее на их добычу тратится время.

Исходя из этого, задача оптимизации сводится к нахождению набора соседних блоков, сумма весов которых будет иметь максимально большое возможное значение. При этом, на такое множество накладывется ряд ограничений, исходящих из необходимости соблюдения допустимых углов наклона стенок карьера.

Обозначим матрицу высот V_{IxJ}, такую, что значение каждого её элемента $v_{i,j}$, $i \in [0,I]$, $j \in [0,J]$ показывает глубину оптимального карьера в каждой точке горизонтальной плоскости.

В таком случае, для учитывания ограничений по максимальному углу наклона стенок карьера ввводится матрица A_{IxJxK}, каждый элемент $a_{i,j,k}$, $i \in [0,I]$, $j \in [0,J]$, $k \in [0,K]$ которой показывает максимальную допустимую разницу высот между элементом $v_{i,j}$, и соседними с ним элементами $v_{i+1,j}$, $v_{i-1,j}$, $v_{i,j+1}$, $v_{i,j-1}$, $v_{i+1,j+1}$, $v_{i+1,j-1}$, $v_{i-1,j+1}$ и $v_{i-1,j-1}$.

В исходной модели, как правило, ограничения на углы наклона бортов задаются не в каждой точке месторождения, а в нескольких опорных точках. При этом указываются координаты точки, направление и значение максимального угла наклона. В таком случае, матрица максимальных углов A получается с помощью аппроксимации данных на каждую точку модели месторождения.

2.2 Приведение блочной модели к виду графа

Как уже было сказано в пункте 1.2, при использовании методов теории графов, задача поиска предельных границ карьеров рудных месторождений сводится к задаче нахождения максимального замыкания ориентированного взвешенного графа. В существующих методах граф G = (V,A) имеет количество вершин n, равное количеству блоков блочной модели месторождения, при этом каждому блоку соответствует вершина графа с весом b_i, а количество дуг m определяется прецедентными отношениями между блоками: вершины i и j соединяются направленной дугой с началом в i и концом в j в том случае, если для извлечения блока i необходимо обязательно извлечь блок j. Большое количество вершин и дуг , а также необходимость большого количества просмотров и изменений структуры графа ведут к росту времени, необходимого на вычисление максимального

замыкания этого графа. Рассмотрим другой подход к представлению блочной модели в виде графа.

В работе [12] показано, что наиболее эффективным методом добычи одиночного блока с положительным весом является построение перевёрнутого конуса с вершиной в данном блоке. Конус строится в соответствии с выбранной схемой извлечения блоков (рисунок 1.2). В дальнейшем будет рассматриваться наиболее популярная схема, основанная на извлечении 9 блоков над исходным, для другой схемы метод представления аналогичен.

Целесообразность добычи блока с положительным весом $m_{X,Y,Z}$ определяется исходя из значения суммы конуса с, построенного из данного блока. Сумма конуса имеет вид:

$$S_c = \sum_{i=1}^{Z} m_{x,y,i},\ l = Z - i,\ x \in [X - l;\ X + l], y \in [Y - l; Y + l] \quad (2)$$

В случае, если $S_c > 0$, считается, что прибыль от добычи блока превысит затраты на извлечение данного блока и всего конуса блоков, лежащего над ним и конус вместе с блоком добавляется к решению задачи. Для положительного блока, находящегося в самом верхнем горизонтальном слое необходимости строить конус нет, а сам блок автоматически добавляется к решению задачи. Рассмотрим варианты расположения множества блоков С, которое является любым подмножеством всех положительных блоков месторождения:

Случай 1. Блоки множества С лежат таким образом, что конусы, построенные из этих блоков не пересекаются. В таком случае решение о добавлении конуса к решению принимается отдельно для каждого конуса;

Случай 2. Блоки множества С лежат таким образом, что существуют пересечения между двумя и более конусами множества, либо имеются вложения одного или нескольких конусов в другие. В этом случае

необходимо исследовать связи между конусами с целью нахождения максимальной суммы пересекающихся конусов. Рассмотрим подробнее данный случай. Обозначим множество конусов с положительной суммой как $P \epsilon C$, а множество отрицательных – $N \epsilon C$. В таком случае возможны такие варианты:

1. В множестве C имеются только конусы из подмножества P. В таком случае каждый конус из множества C добавляется к решению задачи;
2. В множестве C имеются конусы из подмножеств P и N. В таком случае, все конусы из подмножества P добавляются к решению задачи, а конусы из подмножества N добавляются в случае, если соблюдается следующее условие: $S_{P_l} < S_{P_l} + S_{N_l}$, где $P_l \in P, N_l \in N$ – любые подмножества. Другими словами, конус с отрицательной суммой добавляется к решению в том случае, если выгоды от добычи конуса из множества N недостаточно, для компенсации извлечения всех блоков конуса, однако блоки, входящие в пересечение с конусами множества P считаются уже добытыми и расходы на их извлечение не учитываются. В дальнейшем удобно представлять такое объединение в виде одной структуры с положительным весом: $P_i = P_l \cup N_l;\ P_i \in P$ (3)
3. В множестве C имеются только конусы из подмножества N. При этом, добыча каждого конуса из этого множества по отдельности считается невыгодной, однако общая сумма блоков некоторых пересекающихся конусов может быть положительной: $S_{N_i} + S_{N_J} > 0;\ N_i, N_j \in N$. В таком случае, обьединение этих конусов можно считать единой структурой $P_i = N_i \cup N_j, P_i \in P$, которую следует добавить к решению задачи.

С учётом всего вышеперечисленного, блочную модель можно представить в виде взвешенного неориентированного графа $G_1 = (V_1, A_1)$, в котором каждая вершина соответствует положительному блоку (основанию конуса) модели и имеет вес, равный сумме блоков, входящих в этот конус. Две вершины i и j соединяются ребром в случае, если конусы c_i и c_j, построенные из этих вершин имеют общие блоки, а вес ребра равен отрицательной сумме общих блоков этих конусов: $w_{i,j} = -S_{c_i \cap c_j}$. В таком случае, вершины с положительным весом будут считаться частью решения задачи, а вершины, имеющие отрицательный вес, будут добавляться к решению в том случае, если сума весов какой-либо комбинации отрицательных вершин и смежных с ними рёбер будет положительной, то есть добыча некоторого объединения отрицательных конусов станет выгодной за счёт наличия у них общих блоков.

При такой форме представления задача поиска предельных границ карьеров сводится к нахождению максимальной суммы замыканий графа. На рисунке 2.2. изображён пример блочной модели месторождения. Зелёным цветом выделены блоки с положительным весом, красной линией – конусы, построенные из этих блоков.

-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	30	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	15	12	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	20	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	37	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	24	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	19	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4

Рис. 2.2. Пример блочной модели и построения конусов

На рисунке 2.3. изображены представления этой блочной модели в виде графов, использующихся в стандартном алгоритме Лерча-Гроссмана и представленном методе.

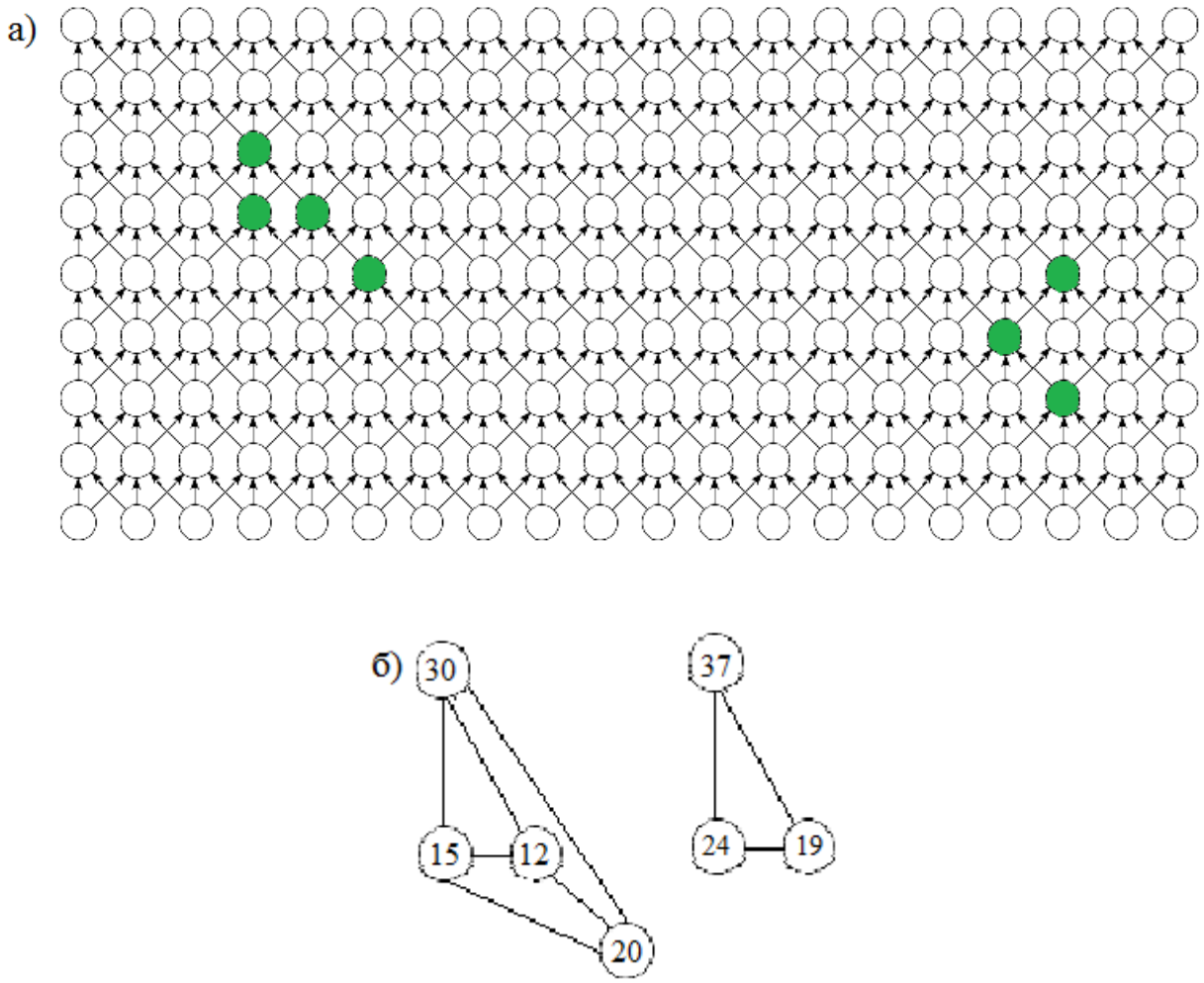

Рис. 2.3. Представление блочной модели в виде графа для алгоритма Лерча-Гроссмана (а) и для разработанного метода (б)

После построения графа, для нахождения решения задачи необходимо к каждому изолированному скоплению вершин применить алгоритм Лерча-Гроссмана, добавив к нему учёт весов рёбер графа при расчёте сильных и слабых вершин. При этом, поскольку изолированные подграфы не связаны между собой, отдельные подграфы можно обрабатывать в параллельном режиме.

2.3. Описание разработанного метода

Процесс расчёта оптимальной оболочки можно разбить на два этапа. На первом этапе строится граф, в виде которого представляется блочная модель. Данный граф является неориентированным, взвешенным, вершины графа соответствуют блокам модели с положительным весом (т.е. блоки, прибыль от добычи которых перевешивает затраты на извлечение, без учёта затрат на извлечение вышележащих блоков). Две вершины соединяются ребром в том случае, если перевёрнутые усечённые конусы, построенные из этих вершин, имеют общие блоки, а вес этого ребра определяется как сумма всех общих блоков. Рассмотрим пошагово построение вышеописанного графа:

Шаг 1. В блочной модели ищется ранее не рассмотренный блок с положительным весом. В случае существования такого блока добавляем к графу новую вершину, соответствующую данному блоку. В противном случае переход к шагу 5.

Шаг 2. В соответствии с выбранной схемой извлечения на модели строится перевёрнутый конус с вершиной в найденном блоке. Конусу присваивается порядковый номер, а суммой конуса на данном шаге считается вес его вершины.

Шаг 3. Для каждого блока, входящего в состав конуса необходимо проделать следующие операции:

1. Вес блока сложить с суммой конуса, полученной на предыдущих шагах;
2. Сохранить информацию о вхождении данного блока в конус с текущим порядковым номером;
3. Проверить, входит ли данный блок в ранее построенные конусы. В случае, если блок является общим для двух и более конусов необходимо:

– При отсутствии ребра, соединяющего вершины графа, которые соответствуют основаниям этих конусов, добавить такое ребро к графу и задать вес ребра, равный значению этого блока в модели.
– При наличии такого ребра заменить его вес сумой веса ребра и значения рассматриваемого блока.

Шаг 4. Возвращение к шагу 1 – поиску нового положительного блока.

Шаг 5. Сохранение полученного графа.

Блок-схема построения графа изображена на рисунке 2.4.

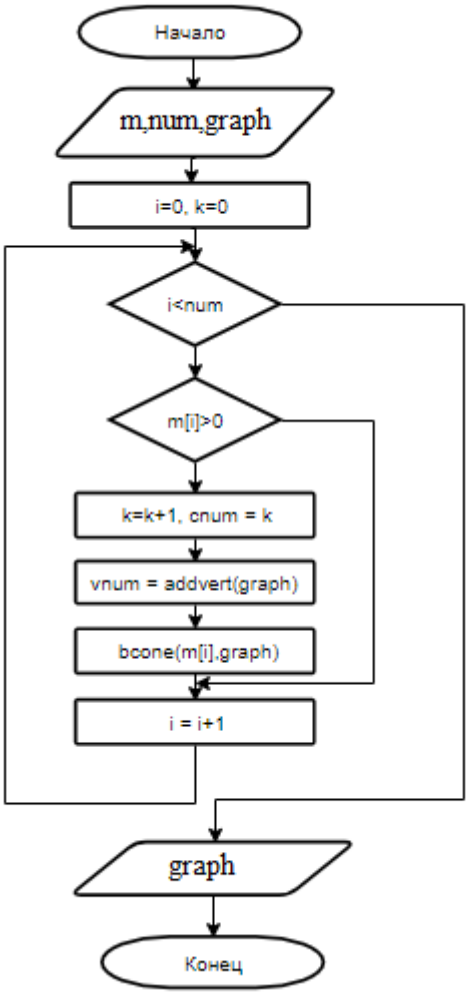

Рис. 2.4. Блок-схема процедуры построения графа месторождения

Функция Addvert добавляет к графу вершину. Так как в качестве представления графа была выбрана матрица смежности, процедура добавления вершины заключается в добавлении новой строки и столбца к матрице.

Функция bcone отвечает за построение конуса, вычисление его суммы, добавление к графу рёбер и пересчёт их весов. Блок-схема функции изображена на рисунке 2.5.

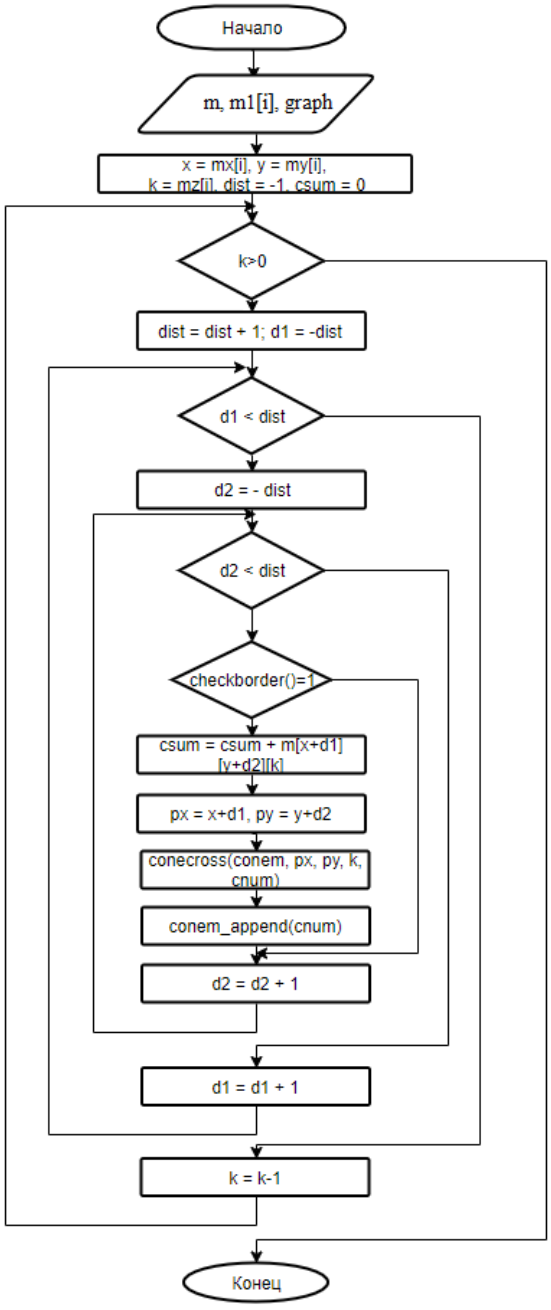

Рис. 2.5. Блок-схема функции bcone

Функция conecross ищет все конусы, в который входит рассматриваемый блок, после чего либо добавляет ребро, либо изменяет вес существующего.

На втором этапе, к каждому изолированному подграфу графа применяется модифицированный алгоритм Лерча-Гроссмана. Блок-схема нахождения максимального замыкания графа представлена на рисунке 2.6.

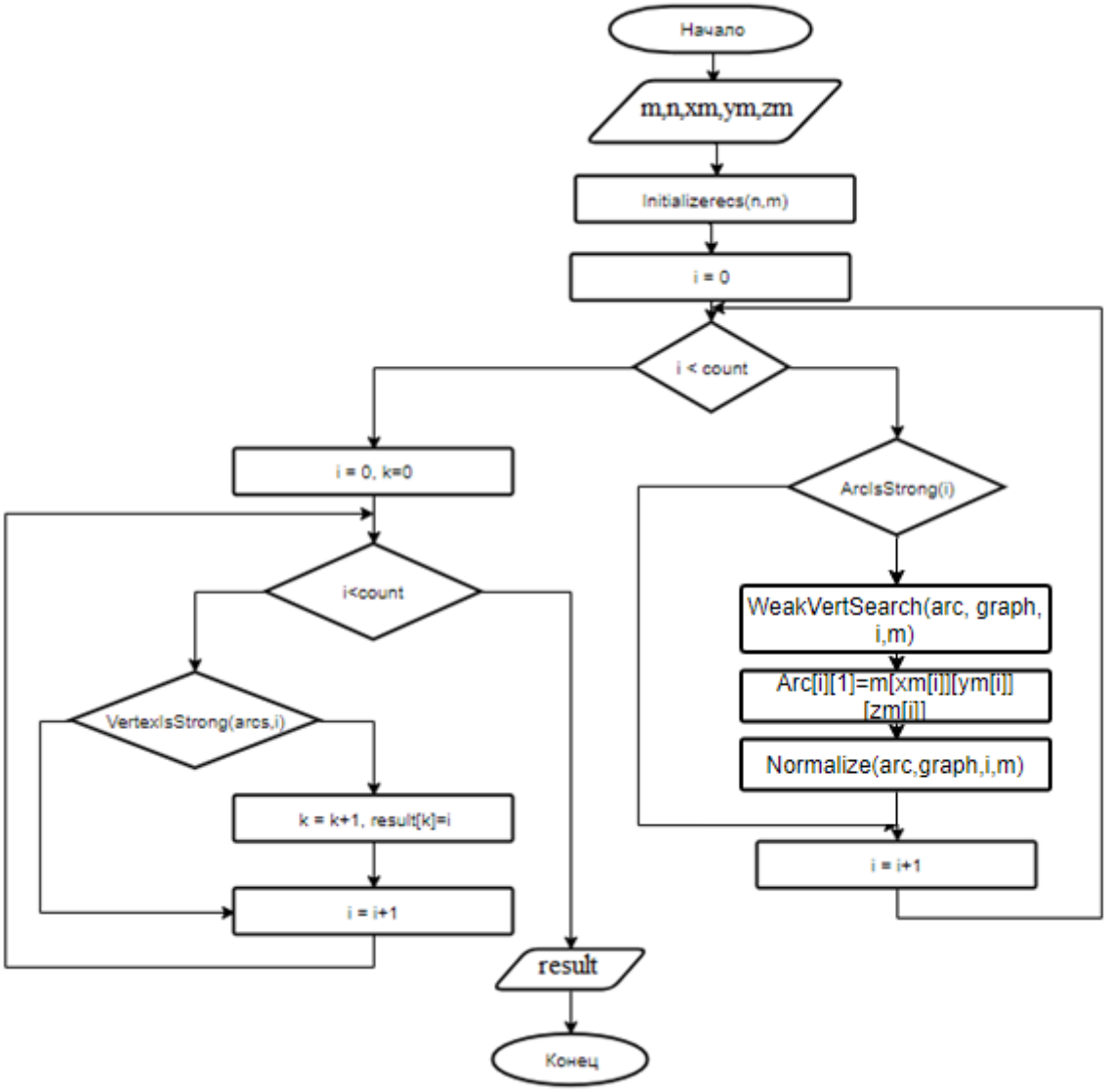

Рис. 2.6. Схема нахождения максимального замыкания графа

Функция ArcIsStrong(i) проверяет, является ли ветвь перебираемой вершины сильной. WeakVertSearch(i,graph,m,ark) отвечает за поиск пары вершин, таких, что существует ребро, соединяющее сильную и слабую

вершину, после чего присваевает началу дуги найденной вершины адрес перебираемой сильной. Функция normalize(i,graph,m,ark) отвечает за нормализацию графа после изменения структуры. Функция VertexIsStrong определяет является ли текущая вершина сильной.

Ассимптотическая оценка временной вычислительной сложности алгоритма

Чтобы иметь возможность оценить время работы метода исключая влияние на время вычисления использование различных аппаратных средств, была проведена оценка сложности разработанного алгоритма. Рассмотрим подробное получение данного значения.

Программу можно разбить на два блока – построение графа (рисунок 2.4) и обсчёт графа (рисунок 2.6). Время выполнения программы складывается из времени выполнения каждого блока. Рассмотрим время выполнения первого блока.

В первом цикле происходит обход каждого из n блоков модели в поиске положительного значения за O(n) времени. Вложенный в него цикл строит перевёрнутый конус с вершиной в данном блоке за время O(n). В цикле построения конуса для каждого блока производится суммирование веса блока с каждым из l записанных в массиве вхождений весов рёбер. Операция суммы имеет сложность O(1) и выполняется для каждой из l_1 записей в массиве вхождений, то есть суммарно операция происходит за время O(l). Выполнение всех операций первого блока занимает $O(n*n*l) = O(n^{2*}l)$ времени.

Во втором блоке происходит обработка графа алгоритмом Лерча-Гроссмана. Полученный граф имеет k вершин и p рёбер. Согласно работе [24], алгоритм Лерча-Гроссмана, применённый к графу G(m,n) имеет вычислительную сложность $O(m*n^2)$. Таким образом, вычисления второго блока занимают $O(p*k^2)$ времени.

Суммарная ассимптотическая оценка сложности – $O(n^{2*}l) + O(p*k^2)$. Поскольку величины p и k незначительны, относительно величин n и l, что

показано на рисунке 2.3, величина $O(n^{2*}l)$ оказывает существенно большее влияние на время вычислений, чем $O(p*k^2)$. В таком случае, окончательная оценка сложности алгоритма $O(n^{2*}l)$.

Для сравнения, в работе [24] приведены вычислительные сложности некоторых алгоритмов. Исходя из этих данных, алгоритм плавающего конуса со сложностью вычислений $O(n^2)$ имеет большее быстродействие, чем представленный метод. Оценка времени алгоритма Лерча-Гроссмана равна $O(m*n^2)$. Поскольку величина m (суммарное количество дуг, ограничивающих крутизну склонов) обычно намного больше, чем величина l (количество всех пересечений конусов положительных блоков), алгоритм Лерча-Гроссмана имеет большую вычислительную сложность, чем описанный в работе метод.

2.4 Параллельный алгоритм поиска предельных границ карьеров

В процессе разработки было выявлено, что в ряде случаев на этапе блоки с положительным весом в модели расположены таким образом, что не существует пересечений некоторых подмножеств конусов, то есть в графе G_1 образуются изолированные вершины и подграфы. Для уменьшения времени вычислений и возможности применения алгоритма с возможностью задействования ресурсов многопроцессорных ЭВМ была разработана параллельная версия поиска решений с одним уровнем параллелизма, в которой каждый подграф обрабатывается независимо отдельным процессом, изображённая на рисунке 2.7.

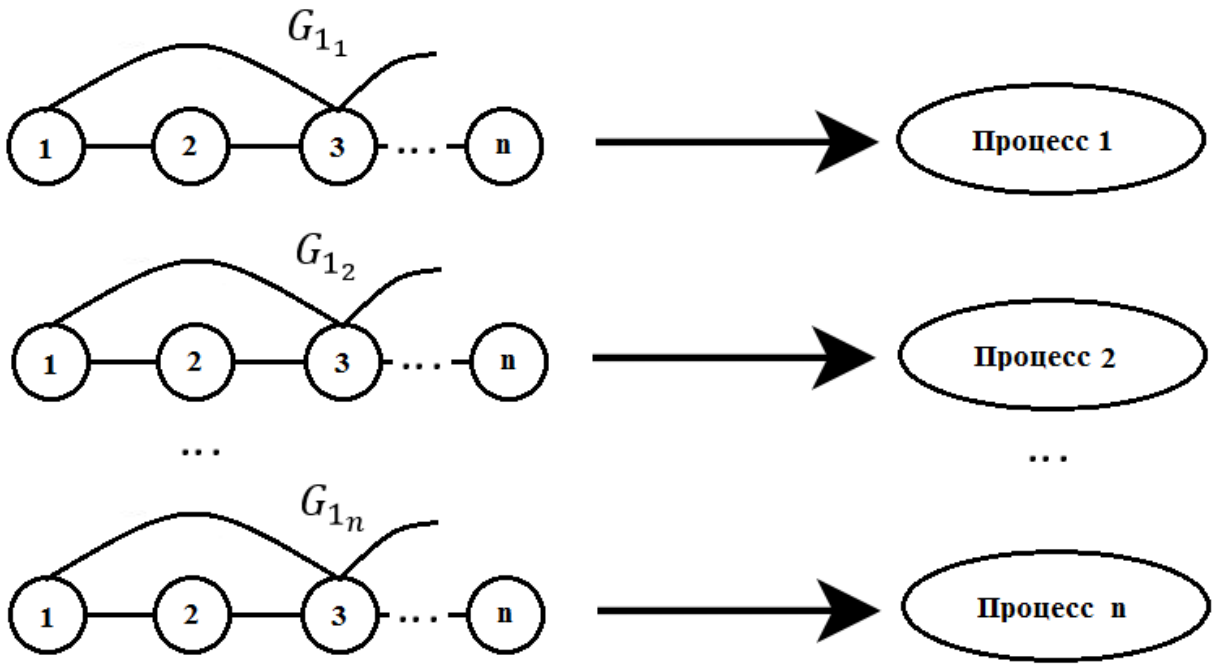

Рис. 2.7. Схема распределения обработки подграфов G_1 между процессами

Ускорение достигается за счёт одновременной обработки нескольких изолированных подграфов. Для обеспечения возможности распараллеливания была необходимо знать количество изолированных подграфов (количество связных компонент), а также список вершин, входящих в каждый подграф, для чего необходимо произвести обход всех вершин графа. С учётом вышеперечисленного была разработана схема параллельного алгоритма поиска замыкания графа G_1, изображённая на рисунке 2.8.

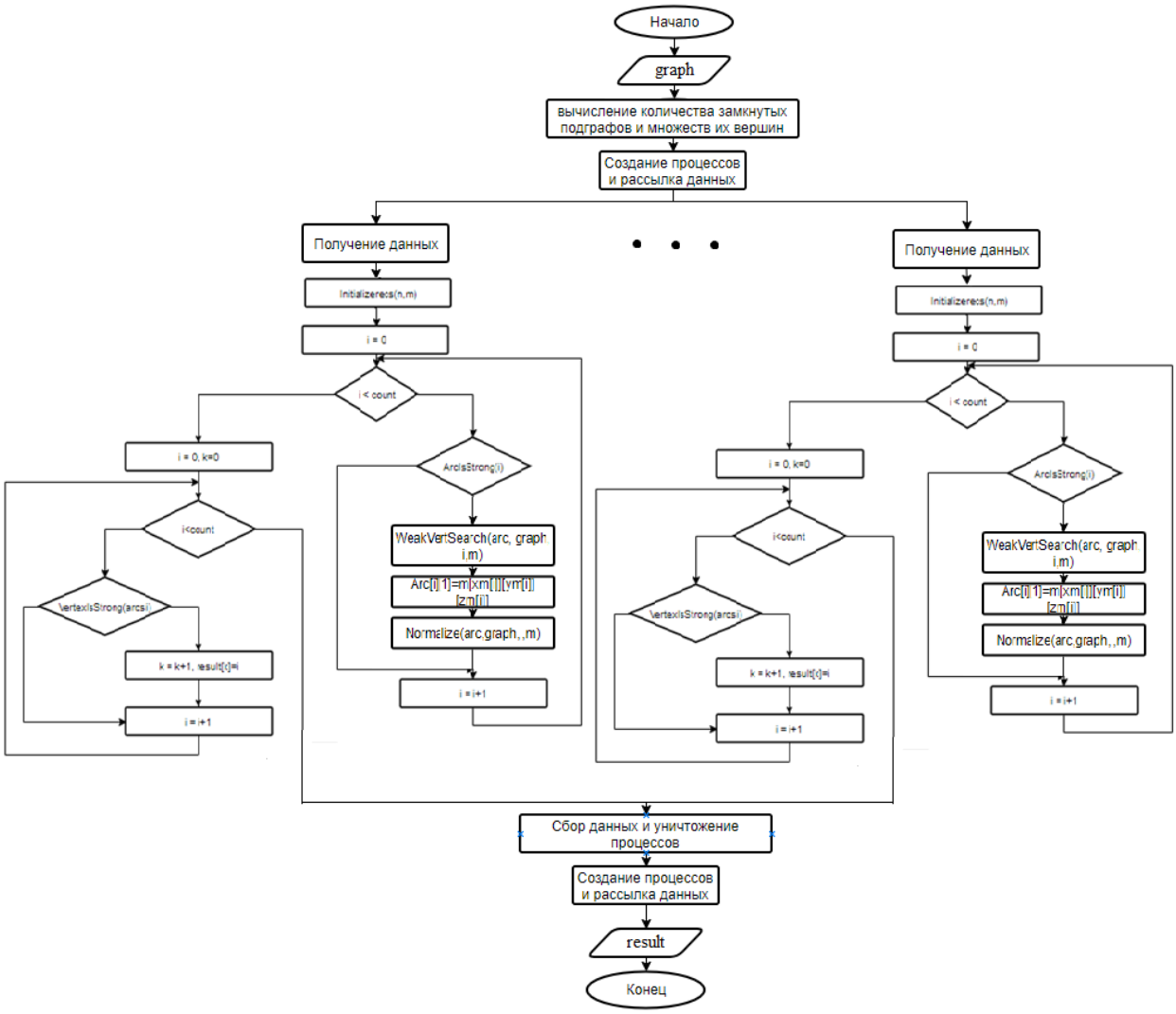

Рис. 2.8. Параллельная версия поиска максимального замыкания графа G_1

ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МЕТОДА ПОИСКА ПРЕДЕЛЬНЫХ ГРАНИЦ КАРЬЕРОВ

3.1 Выбор инструментальных средств разработки

При выборе средства реализации разработанного метода поиска предельных границ карьеров учитывались следующие критерии:

– Удобство и скорость разработки;

– Наличие математических библиотек, содержащих в себе необходимые для реализации метода функции и постоянные;

– Наличие удобных в использовании средств визуализации расчётов;

– Скорость выполнения написанных приложений.

Выбор языка программирования производился между тремя предендентами: C++, Python, Java.

C++ — чрезвычайно мощный язык, содержащий средства создания эффективных программ практически любого назначения, от низкоуровневых утилит и драйверов до сложных программных комплексов самого различного назначения [15]. К преимуществам языка C++ можно отнести: поддержку различных стилей и технологий программирования, включая структурное программирования, объектно-ориентированное программирование, обобщённое и метапрограммирование, возможность работы на низком уровне с памятью и адресами, высокая производительность приложений, написаных на этом языки. Недостатки языка C++, в основном, унаследованы от языка C: синтаксис языка делает весьма вероятным допускание тяжело отлавливаемых ошибок, низкая читаемость кода, плохая поддержка модульности. Всё это приводит к тому, что производительность программиста, использующего C++ сильно падает, по сравнению с другими языками.

Java – объектно-ориентированный интерпретированный язык программирования. Основвными принципами Java считаются: простота языка, безопасность и переносимость кода [1]. Одним из основных преимуществ языка считается огромная библиотека готовых решений, позволяющая свести большую часть работы программиста к поиску готового решения. Другим достоинством считается кросплатформенность программ, написанных на Java – код транслируется в платформонезависимый байт-код, который выполняет виртуальная машина JVM. В свою очередь, JVM может быть установлена почти на любое устройство, что обеспечивает кросплатформенность программ. Основным недостатком этого языка программирования являются ресурсоёмкость и медлительность – используемые технологии автосборки мусора и компиляция "на лету" [22], а также отказ от некоторых опасных механизмов снижает производительность программ, написаных на Java, а использование JVM увеличивает нагрузку на ресурсы компьютера, в частности на ОЗУ. Кроме того, к недостаткам языка можно отнести многословность и громоздкость кода [23].

Python – высокоуровневый язык программирования общего назначения, основной акцент которого сделан на производительность разработчика и читаемость кода [40]. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека содержит большое количество полезных функций, в том числе и математических.

Как и C++, Python поддерживает различные стили и парадигмы программирования, включая структурное, объектно-ориентированное, процедурное, функциональное и аспектно-оринтерованное [4]. Основными чертами архитектуры Python являются – автоматическое распределение памяти, динамическая типизация, средства обработки исключений, поддержка многопоточных вычислений, большое количество встроенных структур данных. К достоинствам Python можно также отнести: переносимость кода между платформами, автоматическую генерацию документации на модули и возможность написания самодокументированных

программ, простой и удобный синтаксис, а также большое количество библиотек [3].

Также, для всех основных поддерживаемых языком платформ, Python имеет поддержку основных технологий, характерных для каждой конкретной платформы. Кроме того, существуют различные реализации языка, такие как PyPy, Cython, Jython и другие, позволяющие использовать основные преимущества других языков в программировании приложений на Python.

Все из перечисленных претендентов в основном удовлетворяют поставленным требованиям. Тем не менее, выбор был сделан в пользу языка программирования Python, поскольку основные преимущества Java – безопасность и кросплатформенность приложений не играют решающей роли в данном проекте, а многословный код существенно снижает производительность разработчика, с другой стороны, несмотря на то, что приложение, написанное на C++, возможно будет работать быстрее, использование этого языка может существенно замедлить разработку приложения, за счёт возможностей появления большого количество ошибок в коде [34]. Из рассмотренных языков программирования, Python больше всего подходит для написания тестового приложения, предназначенного для проверки работоспособности алгоритма и проведения вычислительных экспериментов.

Разрабатываемую программу можно разбить на несколько функциональных блоков, отвечающих за реализацию основных этапов поиска решения задачи нахождения предельных границ карьеров рудных месторождений:

- Блок считывания исходных данных;
- Блок построения модели месторождения;
- Реализация построения графа модели:
    1. Поиск не рассмотренных блоков с положительным весом в модели;

2. Добавление к графу вершин, соответствующим найденным блокам;
3. Построение перевёрнутых конусов с вершинами в найденных блоках;
4. Добавление к графу рёбер в соответствии с найденными пересечениями конусов;
5. Вычисление весов рёбер графа;

– Реализация нахождения суммы максимальных замыканий множеств изолированных подграфов:
    1. Формирование первоначальных графов деревьев с рёбрами, соединяющими фиктивный корень со всеми вершинами графов;
    2. Поиск множеств сильных ветвей в каждом дереве, построенном на предыдущем шаге и добавление их к решению;
– Процедура сохранения полученных результатов;
– Процедура отображения результатов в графическом виде.

3.1. Реализация последовательной версии метода

Считывание исходных данных

Для хранения данных о блочной модели месторождения используются текстовый файл с определённой структурой. Первая строка файла хранит три целых числа – размер трёхмерной модели, а также количество блоков, для которых записаны значения. Во второй строке записана стоимость извлечения пустого блока, то есть блока с нулевым содержанием полезного материала. С целью экономии информация о таких блоках не хранится. Каждая следующая строка содержит в себе три числа – координаты блока в трёхмерной модели и его стоимость, с учётом содержания полезной нагрузки и бесполезной породы. На рисунке 3.1. приведен фрагмент файла с исходными данными.

500 500 500 148205

-5

0 0 200 65

0 0 201 33

0 0 202 85

0 0 203 50

…

500 500 482 -20

Рисунок 3.1. - фрагмент файла с исходными данными

При использовании готовых моделей из библиотеки Minelib [9], необходимо конвертировать исходные данные в описанный формат, поскольку исходные данные с этого сайта расположены в двух файлах форматов .blocks и .upit, кроме того, содержат избыточную для данной задачи информацию. В файле формата .blocks в каждой строке первое целое число – номер блока, затем три числа – координаты блока в модели, все остальные данные в решении задачи поиска предельных границ карьеров не используются. В файле формата .upit, в первой строке содержится имя модели, во второй – тип файла, в третьей – количество блоков. Затем идёт блок данных, в каждой строке которого первое число – порядковый номер блока, соответствующий номеру в файле .blocks, второе число – фактическая ценность блока с учётом содержания в нём полезного сырья и породы. Кроме того, в файлах не указывается размер модели и стоимость извлечения пустого блока, эти данные необходимо вводить вручную. На рисунке 3.2 представлен пример файлов данной библиотеки.

```
0 0 1 20 FRWS 0.077065843 2192.93 1.02 -2236.7886 -36475.56586 0
1 1 1 15 FROR 1.375353107 5664 1.04 -5890.56 24829.116 1
2 1 1 16 OXOR 0.913001543 5184 1.03 -5339.52 34347.213 0
3 1 1 17 OXOR 0.60628858 5184 1.03 -5339.52 6743.223 0
4 1 1 18 OXOR 0.208912037 5184 1.03 -5339.52 -29020.437 0
5 1 1 19 OXOR 0.186089559 1015.64 1.02 -1035.9528 -6077.8936 0
```

а)

```
NAME: Newman1
TYPE: UPIT
NBLOCKS: 1060
OBJECTIVE_FUNCTION:
0 -2236.7886
1 24829.116
2 34347.213
3 6743.223
4 -5339.52
5 -1035.9528
```

б)

Рисунок 3.2. Фрагменты файлов .blocks (а) и .upit (б) исходных данных библиотеки minelib

Преобразование данных библиотеки к формату, используемому разаботанным приложением выполняет функция convert, описанная в листинге 3.1.:

Листинг 3.1. Функция конвертации.

```
def convert():
    print("converting files...")
    upit = open('newman1.upit','r')
    blocks = open('newman1.blocks','r')
    inp = open('input.txt','w')
    upit.readline()
    upit.readline()
    str = upit.readline()
    arr = str.split(' ')
    nblock = int(arr[1])
    upit.readline()
```

После чего происходит построчное считывание данных из файлов .blocks и .upit и запись в файл inp.txt, изображённое в листинге 3.2.:

Листинг 3.2. Запись данных в описанном формате.

```
for i in range (nblock):
    ustr = upit.readline()
    bstr = blocks.readline()
    uarr = ustr.split(' ')
    barr = bstr.split(' ')
    str = barr[1]+' '+barr[2]+ ' '+barr[3]+ ' '+uarr[1]
    inp.writeline(str)
upit.close()
blocks.close()
```

Предварительная подготовка исходных данных

После конвертации, либо, если в качестве исходных данных используется файл с используемой приложением структурой данных, происходит построчное считывание данных и построение трёхмерной модели в виде трёхмерной матрицы чисел. За это отвечает функция mbuild. После инициализации матрицы значениями пустых блоков, происходит считывание данных из файла и замена значений соответствующих блокам данными из файла, изображённая в листинге 3.3:

Листинг 3.3. Построение модели месторождения

```
for i in range (nblock):
    str = inp.readline()
    x = int(str[0])
    y = int(str[1])
    z = int(str[2])
    val = float(str[3])
    m[x][y][z] = val
```

Построение графа модели

Для хранения графа в памяти ЭВМ была выбрана матрица смежности, в которой значение на пересечении i-го столбца и j-ой строки соответствует весу ребра, связывающего вершины i и j. Поскольку ребро с нулевым весом не влияет на результат, считается, что такого ребра не существует. В процессе построения, в матрице блочной модели осуществляется поиск блока с положительным весом, описанный в листинге 3.4.

Листинг 3.4. Поиск положительных блоков и добавление вершины к графу.

```
for i in range (sizex):
   for j in range (sizey):
      for k in range (sizez):
         if m[i][j][k] > 0:
            cnum +=1
            addvert(cnum)
            csum = bcone(i,j,k,cnum)
```

```
        cssum.append(csum)
```

Функция advert, описанная в листинге 3.5. отвечает за добавление вершины к графу.

Листинг 3.5. Функция добавления вершины.

```
def addvert(num):
    graph.append([])
    for i in range (num):
        graph[num].append(0)
        graph[i].append(0)
    graph[num].append(0)
```

В функции bcone, описанной в листинге 3.6, происходит построение конуса из найденной вершины, подсчёт суммы конуса и изменение графа.

Листинг 3.6. Подсчёт суммы конуса и изменение весов графа.

```
def bcone(x,y,z,cnum):
    csum = 0
    dist = -1
      while z>=0:
       dist +=1
       d1 = -dist
       while d1<=dist:
          d2 = -dist
          while d2<=dist:
```

После чего веса рёбер графа, которые соответствуют каждому пересечению конусов, содержащему рассматриваемый блок, изменяются на значение данного блока, а информация о вхождении блока в пересечение сохраняется в conematr. Описание процедуры представлено в листинге 3.7.

Листинг 3.7. Добавление информации о вхождении блока в пересечение.

```
for l in range (len(conematr[x+d1][y+d2][z])):
    graph[cnum][conematr[x+d1][y+d2][z][l]]+=matr[x+d1][y+d2][z]
conematr[x+d1][y+d2][z].append(cnum)
```

Поиск решения задачи на графе

Как было описано ранее, решение задачи поиска предельных границ карьеров сводится к поиску максимального замыкания графа G_1. Поскольку на практике в большинстве случаев граф G_1 состоит из нескольких изолированных подграфов, каждый из них можно рассматривать как отдельный граф и находить каждое замыкание отдельно. Это позволит в дальнейшем расчитывать замыкание каждого подграфа на отдельном процессоре, за счёт чего будет достигаться параллельное выполнение программы. В таком случае, сумма максимальных замыканий подграфов будет решением задачи.

Инициализация записей происходит в функции Initialazerecs(m,m). Затем происходит инициализация счётчика, после чего происходит перебор вершин графа. Функция WeakVertSearch производит поиск слабой вершины в подграфе, после чего изменяет структуру графа так, что началу ветви найденной слабой вершины приваивается адрес текущей сильной вершины, после чего происходит переход к новой вершине. Перечисленные процедуры описаны в листинге 3.8.

Листинг 3.8. Изменение структуры графа.

```
i = 0;
while i<count:
   if (ArcIsStrong(i)==True):
      WeakVertSearch(i,graph,m,arc)
      Weight = arc[i][1] + m[xm[i]ym[i]zm[i]]
      Normalize(i,graph,m,arc)
   i+=1
```

Затем происходит поиск множества сильных вершин графа. Все сильные вершины найденные ранее добавляются к решению, код представлен в листинге 3.9:

Листинг 3.9. Формирование конечного решения.

```
i = 0;
while i<count:
   if (VertexIsStrong(arc,i)==True):
      result.append(i)
   i+=1
```

Сохранение результата

Результатом работы алгоритма является двухмерная матрица размера m на n, которая соответствует горизонтальной плоскости модели. Значение матрицы в координатах (x,y) показывает максимальную глубину оптимального карьера в этой точке месторождения. Кроме того, для проведения вычислительных экспериментов также записывается информация о времени работы программы и суммарная выгода от добычи всех блоков оптимального карьера.

Эта информация сохраняется в текстовый файл посредством функции ResSave(). Для получения оптимальной формы, из каждой вершины решения необходимо построить конус в соответствии с выбранной моделью. Поскольку нам необходима только максимальная глубина в каждой точке, для нахождения результата достаточно найти лишь крайние точки конуса по координатам (x,y) в каждом горизонтальном слое. Сначала производится поиск сильной вершины в списке результатов. Затем из найденных точек функцией buildouttercone ищется множество крайних точек конуса в каждом слое, после чего в матрицу результата заносится глубина этих точек. Описанные процедуры представлены в листинге 3.10.

Листинг 3.10. Поиск глубины оптимального карьера в горизонтальных координатах модели.

```
l = 0
   for l in range result():
        i = m1[l][0], j = m1[l][1], k = m1[l][2]
if k >mres[i][j]:
         mres[i][j]=k
         buildouttercone(m,mres,i,j,k)
```

После чего в файл производится запись матрицы результата mres, описанная в листинге 3.11.

Листинг 3.11. Сохранение результатов.

```
for i in range (sizex):
      j = 0
      for j in range(sizey):
         fres.write(str(mres[i][j])+' ')
```

```
        fres.write('\n')
    fres.close()
```

Визуализация результата

Для представления результата в графическом виде была использована интерактивная среда программирования MATLAB. С помощью MATLAB можно анализировать данные, разрабатывать алгоритмы, создавать модели и приложения. Язык, инструментарий и встроенные математические функции предоставляют возможность исследовать различные подходы и получать решение быстрее, чем с использованием электронных таблиц или традиционных языков программирования.

Язык программирования MATLAB был специально разработан для наиболее быстрого и удобного решения широкого спектра научных и прикладных задач, таких, как: моделирование объектов и разработка систем управления, проектирование коммуникационных систем, обработка сигналов и изображений и другие.

Ядро MATLAB предоставляет простые инструменты для работы с различными типами данных и структурами. Кроме того, имеются возможности для быстрой и простой графической визуализации данных.

Для получения визуального результата из файла решений, полученного на этапе сохранения, считывается матрица глубин карьера, после чего она умножается на -1 и к ней применяется функция surf(). Данная функция строит гладкую поверхность по точкам матрицы, что позволяет получить поверхность оптимального карьера.

3.3 Реализация параллельной версии медота

В настоящее время хорошо развиты и активно применяются при разработке различных прикладных и исследовательских программ в случаях, когда неоходимо увеличить скорость вычислений.

К наиболее распространённым технологиям параллельных вычислений можно отнести:

- OpenMP – открытый стандарт распараллеливания програм. Предоставляет директивы компилятора, библиотеку процедур и переменный окружения, необходимые для программирования многопоточных приложений для многопроцессорных систем с общей памятью [4];
- MPI (Message Passing Inrerface) – программный интерфейс, который реализует возможность обмена сообщениями между процессами, выполняющими одну задачу [6]. Используется для разработки программ, предназначенных для работы на кластерах и суперкомпьютерах и ориентирован на системы с распределённой памятью;
- CUDA (Compute Unified Device Architecture) – программно-аппаратная архитектура параллельных вычислений, расчитанная на использование графических процессоров для ускорения вычислений [23].

Каждая из перечисленных технологий предназначена для использования с различным аппаратным обеспечением и решения различных задач. Для решения задачи, поставленной в данной работе подходит технология OpenMP. К сожалению, в языке программирования Python не присутствует поддержка данной технологии, однако её общие принципы реализуются в технологиях работы с процессами Python Multiprocessing и потоками Python Threading.

Модуль threading используется в языке программирования Python для достижения параллельности с использованием потоков. Потоки позволяют приложениям выполнять в одно и то же время множество задач. Однако использование нескольких потоков в одно время создаёт проблему разграничения доступа к общим данным. Решением этой проблемы является использование блокировки. Однако применение блокировки порождает проблему deadlock, когда потоки взаимно блокируют общие данные, при этом каждый поток ждёт, пока другой снимет блокировку. Получается неопределённая ситуация с бесконечным ожиданием [32].

В языке Python для обхода этих проблем интерпретатором используется внутренний глобальный блокировщик (GIL), который позволяет одновременно выполняться только одному потоку [19]. Это полностью нейтрализует преимущества реализации приложения для многоядерной архитектуры процессоров при использовании модуля threading. Исходя из этого, в данном случае использование этого расширения не является целесообразным.

Модуль multiprocessing был добавлен в язык Python в версии 2.6. Это расширение предоставляет разработчику программный интерфейс для создания процессов, схожий с модулем threading. Пакет python multiprocessing предоставляет разработчику возможность обхода ограничений глобального блокировщика за счёт использование нескольких процессов, вместо потоков. Это, в свою очередь, даёт возможность эффективно использовать ресурсы нескольких процессоров на одной машине. Расширение поддерживает Windows и Unix.

При использовании модуля multiprocessing, под параллельным выполнение программы подразумевается одновременное выполнение множества процессов. Процессы могут выполняться как на нескольких вычислителях, так и на одном. В случае, когда на один процессор приходится более одного процесса, выполнение процессов осуществляется в режиме разделения процессорного времени.

Данная технология использует модель SPMP – single program multiple processes [30]. Это означает, что каждый процесс параллельной программы порождается на основе одного и того же программного кода, который должен быть доступен в тот момент, когда программа запускается на используемых вычислителях.

Исходя из вышеперечисленного, для реализации параллельной версии метода поиска предельных границ был использован модуль python multiprocessing.

Для работы параллельного алгоритма необходимо найти все изолированные подграфы графа G, то есть найти число компонент связности графа. Для этой цели был использован последовательный обход вершин в ширину[hz].

В ходе выполнения работы было отмечено, что считывание исходных данных из файла занимает большое количество времени. Поскольку гарантируется, что в исходном файле о блоке с определёнными координатами будет содержаться только одна запись, то есть при создании трёхмерной модели исключена возможность использования одной ячейки несколькими процессорами, был сделан вывод, что для построения матрицы модели необходимо также использовать технологии параллельного программирования.

Таким образом, в последовательной части программы будут выполняться следующие действия:

- Приведение стуктуры исходных данных к описанному типу (если необходимо);
- Инициализация переменных и структур данных;
- Поиск положительных блоков в модели, построение конусов;
- Построение графа блочной модели;
- Поиск количества связных компонент графа;
- Сохранение данных;

– Визуализация данных.

На параллельную часть программы накладываются следующие функции:

– Считывание исходных данных из файла;

– Построение трёхмерной модели по исходным данным;

– Поиск максимальных замыканий подграфов;

Схема параллельной версии считывания данных представлена на рисунке 15:

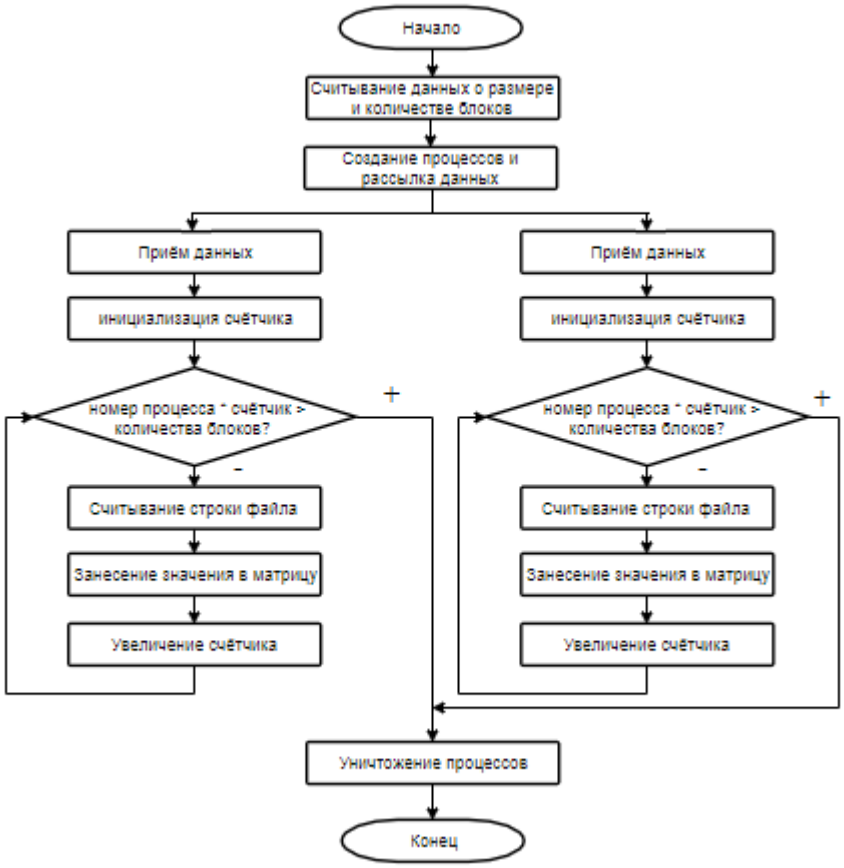

Рисунок 3.3. Схема выполнения паралельного считывания данных

Для создания и управления процессами был использован класс pool, создающий пул процессов и функция map, позволяющая использовать некоторую функцию к каждому элементу списка. Такой подход значительно

упрощает работу с процессами, убирая необходимость отдельно создавать каждый процесс и вручную разделять данные между ними.

Для обмена данными между процессами в технологии используются два типа коммуникации: посредством классов Queue и Pipe. Класс очередей Queue является безопасным для потоков и процессов, поскольку гарантирует, что значение, стоящее на выходе очереди может быть использовано только один раз для всех процессов. Функция Pipe возвращает пару объектов, соединённых между собой. Эти объекты представляют собой концы трубы, каждый из них имеет методы отправки и получения данных. Данные, которые отправлены одним объектом, могут быть получены другим и наоборот. Это исключает возможность повреждения данных в том случае, если к каждому объекту может обращаться не более одного потока.

Модуль multiprocessing имеет все те же методы синхронизации, что и модуль threading:

- Замки. Простейшие замки могут быть реализованы с помощью объекта типа Lock. Замок может иметь два состояния: он или закрыт, или открыт [25]. Запирание замка происходит функцией acquire(), отпирание происходит вызовом функции release(). Отпирание замка должен производить тот же процесс, что и производил запирание. Узнать текущее состояние замка можно командой locked(), однако после проверки состояние блокировки может измениться до того, как будет выполнена следующая команда. Кроме того существует класс RLock - вариант простой блокировки, которая блокирует поток только в том случае, если блокировка захвачена другим потоком.
- Семафоры – более сложный вариант блокировок. в отличии от замка, семафор содержит счётчик и блокирует поток только тогда, когда количество потоков, пытающихся захватить семафор превышает заданное число. В зависимости от инициализации,

семафор может позволить нескольким потокам получить доступ к одной и той же секции программы одновременно. Для инициализации семафора используется функция BoundedSemaphore(max_connections). Аргумент max_connections задаёт максимальное количество потоков, которое может одновременно выполнять внутренний код. Для уменьшения счётчика используется метод acquire(). В случае, если значение счётчика становится равным 0, семафор заблокирует поток, вызвавший данный метод [21]. Для увеличения счётчика используется метод release(). После увеличения счётчика, захваченный на семафоре поток сможет продолжить работу.

Таким образом, обобщённая схема работы параллельной версии программы имеет вид, представленный на рисунке 3.4:

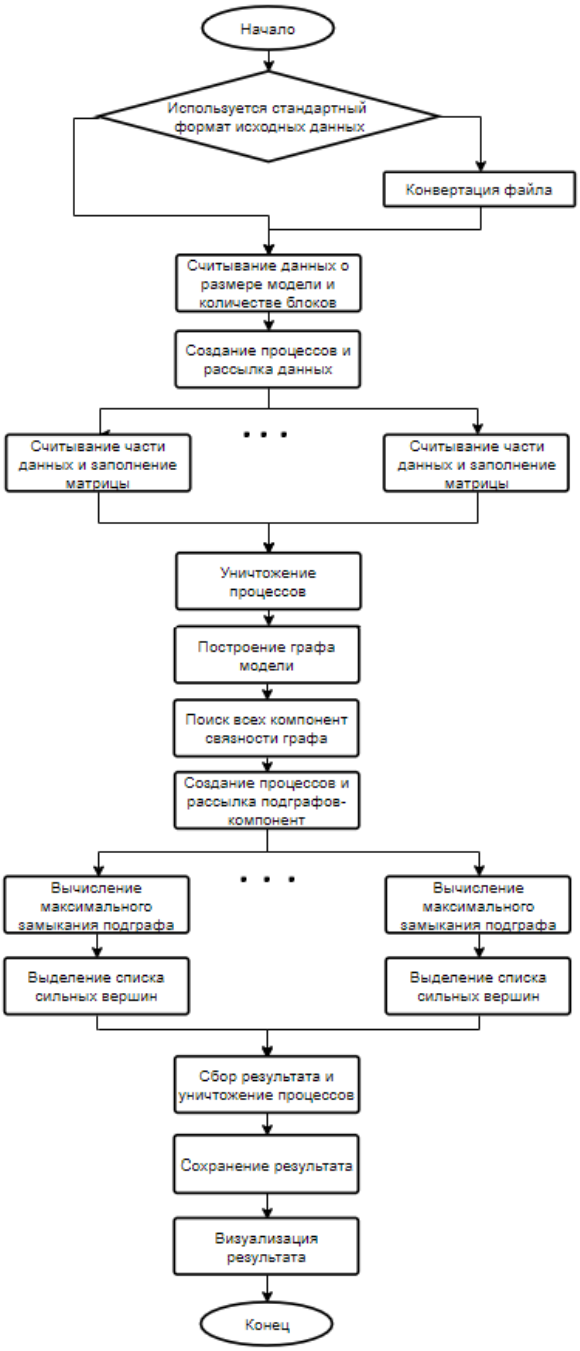

Рис. 3.4. Обобщённая схема работы параллельной версии программы.

ГЛАВА 4. ПРОВЕДЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ ЭКСПЕРИМЕНТОВ

4.1. Технические характеристики аппаратного обеспечения

Для проведения вычислительных экспериментов в качестве оборудования для вычислительных экспериментов был использован персональный компьютер под управлением операционной системы Windows. Данный компьютер оснащён многоядерным центральным процессором семейства Intel Core i5 с поддержкой технологий распараллеливания и гипертрединга. Основные технические характеристики оборудования для вычислений приведены в таблице 4.1.

Таблица 4.1.

Технические характеристики использованного для вычислений оборудования

Характеристика	**Значение**
Семейство процессора	Intel Core i5
Модель процессора	4200U
Частота процессора	1,6 ГГц
Количество ядер	2
Количество потоков	4
Объём ОЗУ	6 Гб
Объём жёсткого диска	512 Гб

4.2 Проведение вычислительных экспериментов

Для проверки работы алгоритма были использованы как реальные модели месторождений, находящиеся в открытом доступе, так и

генерируемые вручную для проверки работоспособности с различными вариантами расположения блоков.

Проверка работы последовательной версии алгоритма

Для проверки работоспособности алгоритма и адекватности получаемых результатов, было произведено вычисление нескольких моделей из библиотеки Minelib [9]. Результаты работы сравнивались с результатами вычисления этих моделей алгоритмами Лерча-Гроссмана и плавающего конуса, а также приведённое на сайте эталонное значение решения задачи. В качестве примера приведены результаты обработки модели newman1, изображение которой представлено на рисунке 17, размером 50x50x25 блоков различными методами.

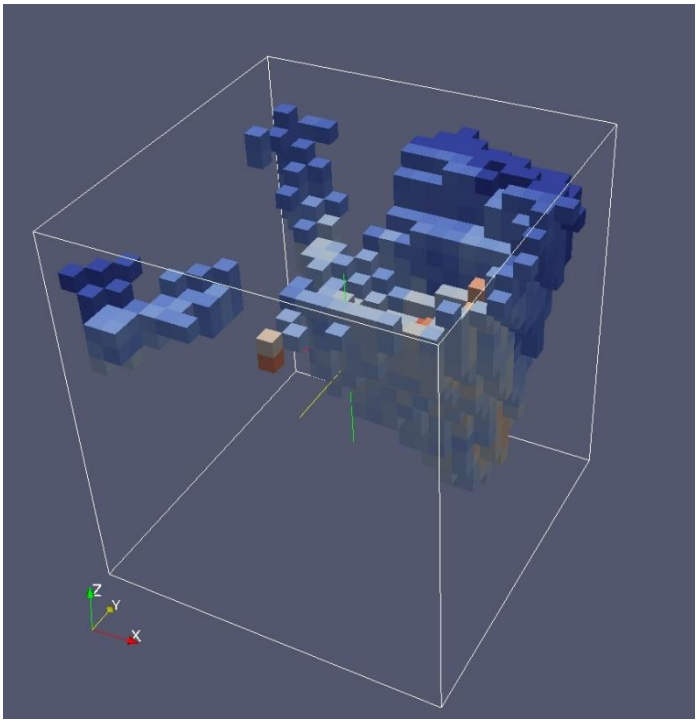

Рис. 4.1. Блочная модель newman1

В таблице 4.2 содержится информация о времени расчёта модели разными методами, а также полученное значение суммы оптимального карьера.

Таблица 4.2.

Данные вычислительных экспериментов над моделью newman1

Характеристика / **Метод**	Время вычислений (сек)	Значение максимальной оболочки	% от эталонного значения
Плавающий конус	601	24179282	92,69
Алгоритм Лерча-Гроссмана	2205	26086899	100
Разработанный метод	1410	26086899	100
Эталонное значение	-	26086899	

Как видно из результатов, разработанный метод не уступает алгоритму Лерча-Гроссмана по качеству найденного результата, однако превосходит его по времени. Алгоритм плавающего конуса работает гораздо быстрее, чем другие, однако не находит оптимальную оболочку карьера в данном случае.

На рисунке 4.2. приведён пример графического отображения полученного результата в среде Matlab. Градацией цветов обозначена глубина карьера в данной точке: синим цветом выделены наиболее глубокие участки, красным – наиболее мелкие.

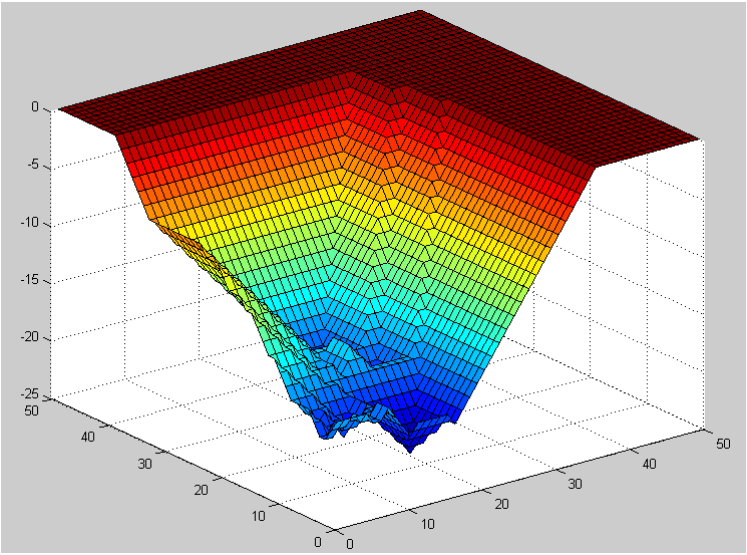

Рис. 4.2. Визуализация результатов поиска оптимальной оболочки карьера модели newman1

Проверка параллельной версии алгоритма

Проверка работы параллельной версии метода в рамках вычислительных экспериментов проводилась на процессоре с двумя физичискими ядрами и поддержкой до четырёх потоков благодаря функции гипертрединга. Цель эксперимента – проверить, насколько меняется время вычисления в зависимости от количества процессов. Кроме того, для проведения вычислений были использованы наборы данных, которые при построении графа модели давали различное количество компонент связности. Модели были разработаны таким образом, что скопления блоков с положительным весом были расположены на таком расстоянии, чтобы суммарные конусные структуры извлечений этих блоков не пересекались. Размеры всех моделей – 50x50x15. Кроме того, для объективной оценки масштабируемости параллельной реализации метода было расчитано ускорение в зависимости от количества используемых процессов по формуле 4.

$$S = \frac{T_1}{T_n}, \qquad (4)$$

где T_1 – время выполнения программы одним процессом, T_n – время выполнения n процессами.

Результаты обработки модели с количеством компонент связности равным шести приведены в таблице 4.3.

Таблица 4.3.

Результаты обработки первой модели

Количество потоков	1	2	4
Время вычислений (сек)	616	409	299
Коэффициент ускорения	1	1,506	2,060

В таблице 4.4. приведены результаты обработки модели с количеством компонент связности, равным двум.

Таблица 4.4.

Результаты обработки второй модели

Количество потоков	1	2	4
Время вычислений (сек)	600	427	387
Коэффициент ускорения	1	1,405	1,550

В случае, когда граф модели G_1 имеет одну компоненту связности, является связным и блок поиска замыкания графа полностью выполняется одним потоком, то есть в последовательном режиме. Тем не менее, считывание данных происходит с задействованием максимального числа процессов, что даёт небольшой прирост в производительности по сравнению с последовательной версией алгоритма. В таблице 4.5 приведены результаты работы последовательного и параллельного алгоритма поиска границ для связного графа G_1.

Таблица 4.5.

Результаты обработки третьей модели

Количество потоков	1	2	4
Время вычислений последовательной версией(сек)	592	-	-
Время вычислений параллельной версией (сек)	603	560	542
Коэффициент ускорения	1	1,077	1,112

На рисунке 4.3. приведён график зависимости времени вычислений от количества процессов для каждой из моделей. Жёлтым цветом обозначено время первой модели, зелёным – второй, красным – третьей, с одной компонентой связности графа G_1.

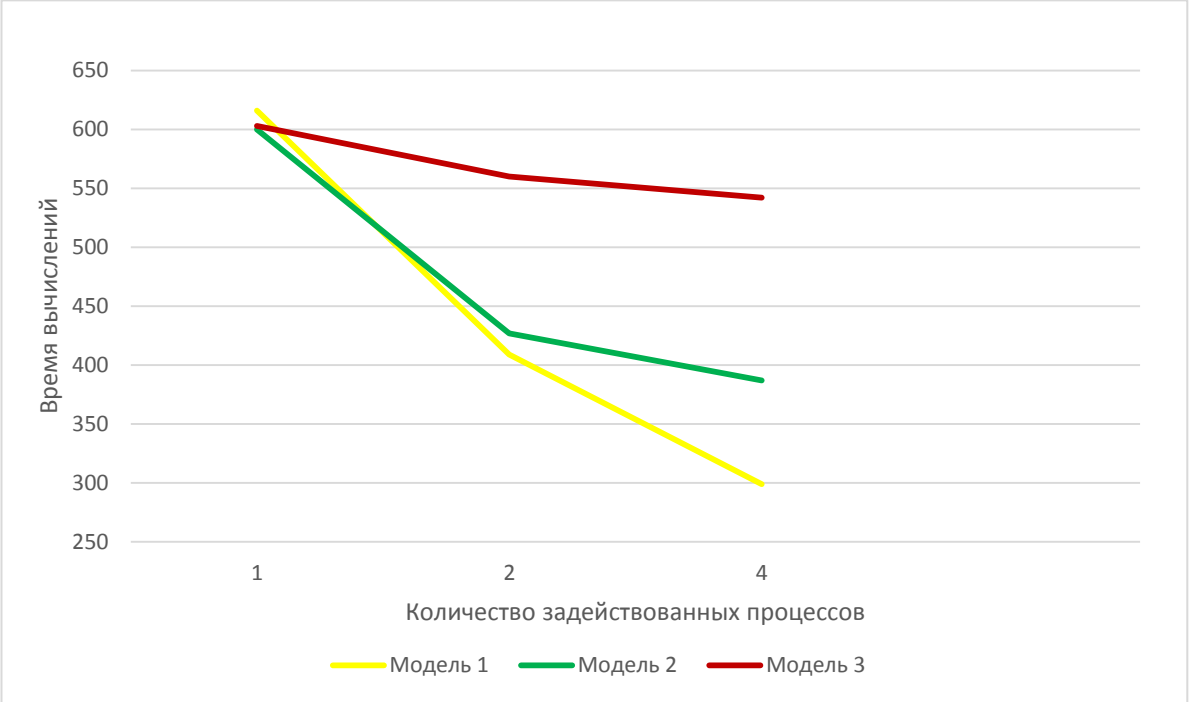

Рис. 4.3. Сравнение времени расчёта трёх моделей разным количеством процессов

На рисунке 4.4. изображён график зависимости коэффициента ускорения вычисления трёх моделей от количества задействованных потоков.

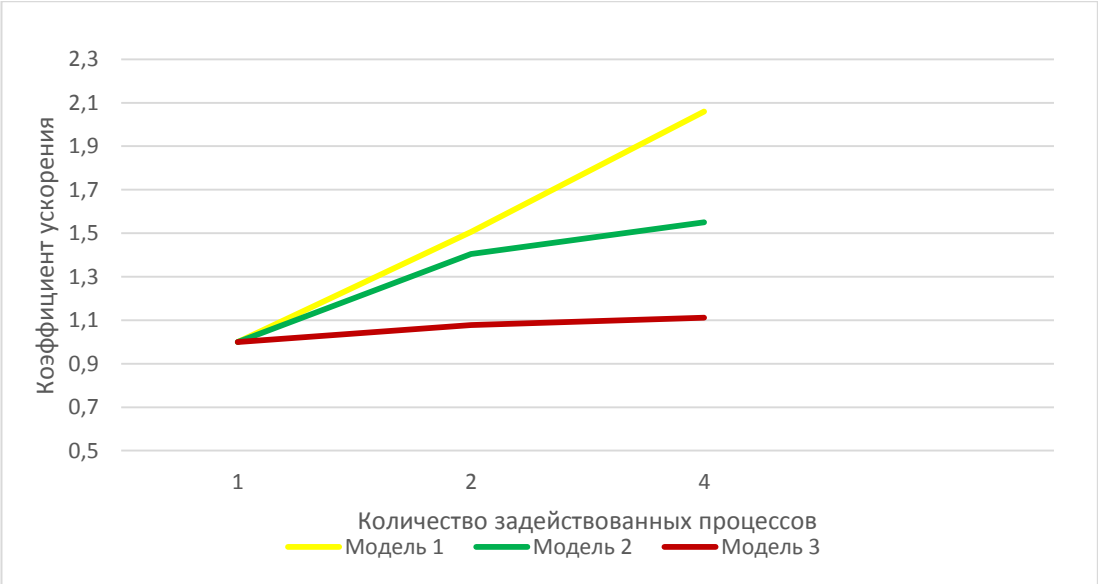

Рисунок 4.4. Сравнение коэффициентов ускорения для трёх моделей на разном количестве потоков

Из результатов эксперимента можно сделать вывод, что параллельная версия алгоритма обеспечивает различный прирост производительности, в зависимости от исходных данных, по сравнению с последовательной версией. Наибольший прирост производительности программа обеспечивает в том случае, если количество связных компонент графа G_1 больше либо равно количеству вычислителей ЭВМ, на которой осуществляется поиск решения задачи.

ЗАКЛЮЧЕНИЕ

В работе рассматривается задача поиска предельных границ карьеров рудных месторождений, проводится анализ существующих путей решения, разработка нового метода решения данной задачи, а также приведена методика распараллеливания разработанного метода для вычислительных структур с общей памятью.

Представленный метод позволяет свести поставленную задачу к поиску некого множества вершин на графе. Методы, основанные на теории графов хорошо показывают себя при решении многих задач, в том числе и поиска предельных границ, поскольку обеспечивают высокую точность результата независимо от вида исходных данных. Алгоритм показывает более приемлемый результат по времени, чем популярный алгоритм Лерча-Гроссмана, при этом не уступая ему в качестве решения. Разработанная параллельная реализация позволяет ещё сильнее сократить затраты времени на обработку модели, что позволяет, при неизменности времени вычислений, увеличить масштаб модели месторождения, что ведёт к повышению точности результатов.

Исходя из вышеперечисленного, цель работы можно считать достигнутой.

В качестве дальнейших возможных путей развития проекта можно предложить следующее:

- Оптимизацию алгоритма путём добавления параллельного послойного построения конусов;
- Разработка метода поиска решения задачи оптимального порядка извлечения блоков;

Результаты вычислительных экспериментов показали высокую эффективность разработанного метода, в сравнении с существующими методами поиска предельных границ карьеров.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Arnold K., Gosling J., Holmes D. The Java programming language. – Addison Wesley Professional, 2005.
2. Beazley D. Understanding the python gil //PyCON Python Conference. Atlanta, Georgia. – 2010.
3. Beazley D. Inside the python GIL (slides) //Python Concurrency Workshop. – 2009.
4. Brian Jones, David Beazley,Python Cookbook, 3rd Edition, O'Reilly Media May 2013, 43 – 45
5. Caccetta L., Hill S. P. An application of branch and cut to open pit mine scheduling //Journal of global optimization. – 2003. – Т. 27. – №. 2-3. – С. 349-365.
6. Chun W. Core python programming. – Prentice Hall Professional, 2001. – Т. 1.
7. Dagdelen K. Open pit optimization-strategies for improving economics of mining projects through mine planning //17th International Mining Congress and Exhibition of Turkey. – 2001. – С. 117-121.
8. Dagum L., Menon R. OpenMP: an industry standard API for shared-memory programming //IEEE computational science and engineering. – 1998. – Т. 5. – №. 1. – С. 46-55.
9. Dimitrakopoulos R., Farrelly C. T., Godoy M. Moving forward from traditional optimization: grade uncertainty and risk effects in open-pit design //Mining Technology. – 2002. – Т. 111. – №. 1. – С. 82-88.
10. Dorit S. Hochbaum. The Pseudoflow Algorithm: A New Algorithm for the Maximum-Flow Problem, Operations Research Volume 56 Issue 4, July 2008 Pages 992-1009
11. Düring B. Sprint driven development: agile methodologies in a distributed open source project (PyPy) //International Conference on Extreme Programming and Agile Processes in Software Engineering. – Springer, Berlin, Heidelberg, 2006. – С. 191-195.

12. E. Elahizeyni, R. Kakaie. A new algorithm for optimum open pit design: Floating cone method III, A. Yousefi, Journal of Mining & Environment,
13. Espinoza D, Goycoolea M, Moreno E, Newman A. Minelib 2011: A library of open pit production scheduling problems. // Ann. Oper. Res., 2013. Vol.206(1), P. 93–114.
14. Grant M., Boyd S., Ye Y. CVX: Matlab software for disciplined convex programming. – 2008.
15. Halterman. Richard L. Fundamentals of. Programming. C++. DRAFT. School of Computing. Southern Adventist University. May 22, 2018
16. Khalokakaie R. Computer-aided optimal open pit design with variable slope angles : дис. – University of Leeds, 1999.
17. Lerchs H. and Grossmann I. F. Optimum design of open pit mines. CIM Bull. , 58, 1965, 47–54. Vol.2, No.2, 2011, 118-125.
18. Mark Lutz, Programming Python, 4th Edition, O'Reilly Media, December 2010, 48 – 49
19. Yegulalp T. M. and Arias J. A. A fast algorithm to solve the ultimate pit limit problem. In Proc. 23rd symposium on the application of computers and operations research in the mineral industries (APCOM) (Littleton, Colorado: AIME, 1992), 391–7..
20. Wen-Mei W. H. GPU computing gems emerald edition. – Elsevier, 2011.
21. Антонов А. С. Параллельное программирование с использованием технологии OpenMP. – 2009.
22. Арнольд К., Гослинг Д. Язык программирования JAVA. – СПб. и др. : Питер-пресс, 1997.
23. Брюс Эккель,Философия Java, Питер, Библиотека программиста, с. 21 – 22
24. Васильев П.В., Михелев В.М., Петров Д.В. Оценка вычислительной сложности алгоритмов оптимизации границ карьеров в системе недропользования // Научные ведомости Белгородского

государственного университета. Серия: Экономика. Информатика. 2015. №19 (216).

25. Васильев П.В., Михелев В.М., Петров Д.В. Применение параллельного алгоритма плавающего конуса для решения задачи поиска предельных границ карьеров // Научные ведомости Белгородского государственного университета. Серия: Экономика. Информатика. 2016. №2 (223).
26. Галустьян Э. Л. Технологические аспекты проблемы оптимизации углов наклона бортов карьеров //Горный информационно-аналитический бюллетень (научно-технический журнал). – 1999. – №. 2.
27. Гергель В. П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем //М.: Издательство Московского университета. – 2010. – С. 534.
28. Дьяконов В., Круглов В. Математические пакеты расширения MATLAB //Справочник. – 2001.
29. Ерёмин Д. И., Ягфарова Н. И., Абишев Д. А. Алгоритм Лерча — Гроссмана и его реализация на центральном и графическом процессорах // Современные тенденции технических наук: материалы III Междунар. науч. конф. (г. Казань, октябрь 2014 г.). — Казань: Бук, 2014. — С. 8-13. — URL https://moluch.ru/conf/tech/archive/123/6334/ (дата обращения: 12.05.2018).
30. Корнеев В.Д. Параллельное программирование в MPI, изд-во СО РАН, Новосибирск, 2000, 213 стр.
31. Левин М. П. Параллельное программирование с использованием OpenMP //М.: Бином. – 2008.
32. Немнюгин С. А. Параллельное программирование для многопроцессорных вычислительных систем. – БХВ-Петербург, 2002.

33. Новожилов М. Г., Куценко В. И., Дриженко А. Ю. Оптимизация параметров высоких уступов при разработке глубоких горизонтов карьеров //Горный журнал. – 1983. – №. 2. – С. 14-18.
34. Павловская Т. А., Щупак Ю. А. C/C++. Программирование на языке высокого уровня. Структурное программирование. – М. и др. : Питер, 2002.
35. Полищук С. З. и др. Прогноз устойчивости и оптимизация параметров бортов глубоких карьеров //Днепропетровск, Полиграфист, 2001 г. 138 с. – 2001. Полищук С. З. и др. Прогноз устойчивости и оптимизация параметров бортов глубоких карьеров //Днепропетровск, Полиграфист, 2001 г. 138 с. – 2001.
36. Россум Г., Дрейк Ф. Л. Д., Откидач Д. С. Язык программирования Python //М.: Вильяме. – 2001.
37. Саканцев М. Г. Оптимизация границ глубоких карьеров цветной металлургии: Автореф. дисс.… канд. техн. наук.-Свердловск, 1983.-19 с. – 1983.
38. Саммерфилд М. Программирование на Python 3. – 2011.
39. Страуструп Б. Язык программирования C++. – Бином, 2011.
40. Сузи Р. А. Язык программирования Python //М.: Бином. Лаборатория знаний. – 2006.
41. Шпаковский Г.И. Реализация параллельных вычислений: MPI, OpenMP, кластеры, грид, многоядерные процессоры, графические процессоры, квантовые компьютеры. Минск. БГУ. 2011 г. - 176 с.

ПРИЛОЖЕНИЕ

Последовательная версия программы

```
from math import *
import time

sizex = 50
sizey = 50
sizez = 50
cssum = [] #cones sums
graph = [[],[]]
matr = [[],[],[]] #input matr
conematr=[[],[],[],[]] #matr with list of cones, current block is included in

#initialization
print("initialazing variables...")
graph.append([])
for i in range (sizex):
    matr.append([])
    conematr.append([])
    for j in range (sizey):
        matr[i].append([])
        conematr[i].append([])
        for k in range (sizez):
            matr[i][j].append(-1)
            conematr[i][j].append([])
            #conematr[i][j][k].append(0)

#file reading
def fileread():
```

```
    print("reading from files...")
    upit = open('newman1.upit','r')
    blocks = open('newman1.blocks','r')
    upit.readline()
    upit.readline()
    str = upit.readline()
    arr = str.split(' ')
    nblock = int(arr[1])
    upit.readline()
    for i in range (nblock):
        ustr = upit.readline()
        bstr = blocks.readline()
        uarr = ustr.split(' ')
        barr = bstr.split(' ')
        x = int(barr[1])
        y = int(barr[2])
        z = int(barr[3])
        val = float(uarr[1])
        matr[x][y][z] = val
    upit.close()
    blocks.close()

def setdata(param):
    if param == "test":
        matr[25][25][10]=200
        matr[30][30][10]=250
        matr[32][32][10]=200
    else:
        fileread()
```

```
#cone calculating
def bcone(x,y,z,cnum):
    csum = 0
    dist = -1
    print("current ore block: %s" %cnum)
    #conematr[x][y][z].append(cnum)
    while z>=0:
        dist +=1
        d1 = -dist
        while d1<=dist:
            d2 = -dist
            while d2<=dist:
                if sizex>=x+d1>=0 and sizey>=y+d2>=0: # checking if current
block is out of borders
                    csum+=matr[x+d1][y+d2][z]
                    for l in range (len(conematr[x+d1][y+d2][z])):

graph[cnum][conematr[x+d1][y+d2][z][l]]+=matr[x+d1][y+d2][z]
conematr[x+d1][y+d2][z].append(cnum)
                    d2+=1
            d1+=1
        z-=1
    return csum

def addvert(num):
    graph.append([])
    for i in range (num):
        graph[num].append(0)
        graph[i].append(0)
```

```
    graph[num].append(0)



launchtype = "test" # "test" for test dataset, anything else for actual dataset
setdata(launchtype)

#building cone for each positive block
print("building cones...")
start_time = time.time()
cnum = -1
for i in range (sizex):
    for j in range (sizey):
        for k in range (sizez):
            if matr[i][j][k] > 0:
                cnum +=1
                addvert(cnum)
                csum = bcone(i,j,k,cnum)
                cssum.append(csum)
print("--- cones built, time elapsed: %s seconds ---" % (time.time() - start_time))
laskfl = input()
```