**Аппроксимация для пакетирования через приоритеты**

Wolfgang BEIN, John NOGA, Jeffrey WIEGLEY

**Аннотация**

В этой работе мы рассматриваем задачу серийного пакетирования на одной машине при среднем значении возможных исходов. Эта задача NP-сложная и нам неизвестны хорошие алгоритмы приближения. Пакетирование имеет широкое применение в производстве, принятии решений и планировании в информационных технологиях.

Мы описываем алгоритм аппроксимации с коэффициентом аппроксимации 2; данный алгоритм является приоритетным алгоритмом, который пакетирует задания в порядке убывания приоритета. Мы также описываем нижнюю границу для коэффициента приближения любого приоритетного алгоритма, и предполагаем, что есть приоритетный алгоритм, который соответствует этой границе. Для поддержки гипотезы использован адаптивный экспериментальный алгоритм. Мы описываем новый линейный временной алгоритм для пакетирования списка задач.

**1 Предыстория и мотивация**

Задачи пакетирования играют важную роль в информационных технологиях. Мы рассматриваем задачу пакетирования, где набор заданий J = {Ji} с временем обработки pi > 0 и весом wi ≥ 0, i = 1,…,n должны быть запланированы на одной машине, и где J должен быть разбит на пакеты B1,…,Br. Все задания в одном пакете выполняются вместе и под временем завершения каждого задания понимается время завершения создание его пакета. Мы предполагаем, что, когда пакет запланирован, требуется установочное время s = 1. Цель состоит в том, чтобы найти график, который минимизирует сумму значений весов завершенных попыток , где Ci обозначает время завершения Ji в заданном графике. Учитывая последовательность заданий, алгоритм пакетирования должен добавить каждое задание Ji в пакет. Более формально, подходящим решением является назначение каждого задания Ji к пакету, i ∈ {1,…,n}.

Например, Рисунок 1 демонстрирует два графика для задачи с пятью заданиями, где время обработки p1 = 3, p2 = 1, p3 = 4, p4 = 2, p5 = 1, а веса w1 = w4 = w5 = 1 и w2 = w3 = 2. Отметим, что значения в круге обозначают сумму весов времён завершения двух изображенных графиков. Например, первый график означает, что C2 = 7, C3 = 5, а C1 = С4 = С5 = 14. Таком образом, сумма весов времён завершения первого графика . Для удобства, мы иногда пишем данные о заданиях в форме {(p1, w1), (p2, w2),…, (pn, wn)}. Таким образом в примере мы бы написали {(3,1), (1,2), (4,2), (2,1), (1,1)}.



Рисунок 1 – Пример пакетирования (s-пакет).

В задаче, рассматриваемой в этой статье, задания выполняются последовательно, поэтому эту задачу более точно называют задачей s-пакета. Отметим, что существует другая версия проблемы, которая здесь не освещалась, где задания пакета выполняются параллельно, она так же известна, как задача p-пакета. В этом случае, длина пакета – это максимальное время обработки его заданий. На Рисунке 2 приведён пример графика p-пакета.



Рисунок 2 – Пример пакетирования (p-пакет)

Обратим внимание, что в примере C1 = C2 = C4 = 3, а C3 = C5 = 8, и, таким образом, сумма весов времён завершения 3(1+1+2)+8(2+1) = 36. Обратите внимание, что эта версия задачи решается алгоритмом O(n logn) из-за [7]. Рассматриваемая здесь задача s-пакета более точно обозначается, как задача в обозначениях . Брукер и Альберс [1] показали, что задача – это задача NP-сложная, что, грубо-говоря, является сокращением от 3-PARTITION.

Существует большой объем работ по задачам пакетирования (см., [2, 3, 6, 7, 9, 12]), также пакетирование имеет широкое применение в производстве (см., [8, 16, 20]), принятии решений (см., [14]) и планировании в информационных технологиях (см., [10]). Более поздняя работа по онлайн-пакетированию связана с задачей подтверждения TCP (Transmission Control Protocol) (см., [4, 11, 13]). Мы также отмечаем, что существует много вариантов задачи, с результатами различной сложности. Для обзора, см. главу 8 из [6]. Тем не менее, задача считается фундаментальной в теории планирования, и это действительно удивительно, что мало что известно об его аппроксимации.

Здесь мы описываем алгоритм аппроксимации для задачи. На самом деле, мы описываем два алгоритма, один называется ПсевдоПакет, а другой КаноническиЛучший. Для алгоритмов приближения, это естественно рассматривать задачи в соответствии с порядком приоритетов . Если задачи пронумерованы так, что , мы говорим, что задачи находятся в каноническом порядке. Алгоритм, который планирует задания в этом порядке называется приоритетным алгоритмом. Оба наших алгоритма аппроксимации являются приоритетными алгоритмами.



Рисунок 3 – Демонстрация пакетирования

Напомним, что качество аппроксимации является мерой с точки зрения его коэффициента аппроксимации ρ: учитывая задачу оптимизации P, мы говорим, что у алгоритма Ap коэффициент аппроксимации ρ, если для каждого экземпляра π ∈ P,

где минимум – это значение минимального решения. Мы показываем, что ПсевдоПакет и КаноническиЛучший имеют коэффициент аппроксимации . Мы также описываем нижнюю границу коэффициента аппроксимации любого приоритетного алгоритма и предпологаем, что КаноническиЛучший соответствует этому ограничению. Эксперименты с адаптивными алгоритмами используются для поддержки гипотезы.

Гораздо более простой вариант задачи – это список, в котором указан порядок заданий, т.е. . Например, на рисунке 3 показаны три графика для задач с пятью заданиями {(3,1)(1,1)(4,1)(2,1)(1,1)}. Значения в круге показывают сумму взвешенных времен завершения графиков слева.

Брукер и Альберс [1] описали линейный алгоритм времени для пакетной обработки списка задач. (Таким образом, для решения задачи достаточно знать порядок заданий в оптимальном решении.) В этой статье мы описываем альтернативный алгоритм. Наш алгоритм использует факт того, что задача может быть сведена к задаче кратчайшего пути, где базовая матрица затрат полностью монотонная матрица и, следовательно, может использовать алгоритм поиска матрицы Лармора и Шибера [15] в качестве подпрограммы. Матрица А называется полностью монотонной, если для всех

i < i’ и j < j’, A[i,j] > A[i, j’] следовательно A[i’, j] > A[i’, j’]; матрица A называется матрицей Монжа если A[i,j] + A[i’, j’] ≤ A[i’,j’] + A[i,j’]. Очевидно, что каждая матрица Монжа является полностью монотонной. Отметим, что линейный алгоритм списка времени используется для реализации КаноническиЛучшего в процессе времени O(n logn).

Наша статья организована следующим образом: в разделе 2 мы описываем приоритетные алгоритмы аппроксимации. Раздел 3 демонстрирует альтернативный алгоритм линейного времени для задачи пакетной обработки списка. В разделе 4 представлена нижняя граница коэффициента аппроксимации любого приоритетного алгоритма. Раздел 5 описывает эксперименты генетического алгоритма. В частности, мы проводим эксперимент с адаптивным алгоритмом, который поддерживает гипотезу о том, что КаноническиЛучший коэффициент аппроксимации соответствует нижней границе. Этот раздел также содержит описание генетического алгоритма для задачи дополненной GAlib, объектно-ориентированной библиотекой Мэтью Уолла [19], разработанная в Массачусетском техническом институте. Мы заканчиваем эту статью открытыми проблемами в разделе 6.

**2 Алгоритмы аппроксимации**

Мы опишем следующую техническую лемму, которая также известна, как «правило Смита» в области планирования.

**Лемма 1**

Даны минимизируется, когда идентично.

**Доказательство:** рассмотрим перестановку τ, которая не является идентичной. Тогда τ имеет инверсию j > i с i непосредственно перед j в τ. Пусть τ’ – перестановка с заменой i и j. У нас есть

fτ′ − fτ = piwi + (pi + pj )wj − pjwj − (pi + pj )wi

= piwj − pjwi

≤ 0,

Поскольку , следовательно .

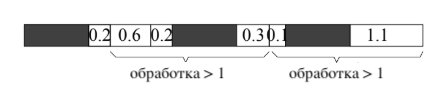
**

Рисунок 4 – ПсевдоПакет для p1 = 0.2, p2 = 0.6, p3 = 0.2, p4 = 0.3, p5 = 0.1, p6 – 1.1

**Лемма 2** Пусть Ci – время завершения оптимального расписания для задачи , и пусть . Тогда мы имеем

**Доказательство:** пусть перестановка σ будет порядком оптимального расписания. Тогда

По Лемме 1 имеем

Теперь мы представляем простой параметризированный алгоритм, ПсевдоПакет, для задачи . ПсевдоПакет сначала переупорядочивает задачи в каноническом порядке. Затем задачи назначаются в пакеты в этом порядке. После получения Ji, наш алгоритм имеет только 2 варианта, а именно назначить ли Ji тому же пакету, что и Ji-1 или нет. Мы используем фразу «Пакеты на шаге i», которая означает, что алгоритм А решает, что Ji является первой задаче в новом пакете, т.е. mi = mi – 1 + 1. Также мы используем фразу «текущий пакет», чтобы обозначит пакет, которому была назначена последняя задача. Затем, когда Ji получен, А должен решить, добавить ли Ji в текущий пакет или «закрыть» текущий пакет и назначить Ji на новый пакет. ПсевдоПакет поддерживает переменная P, которая является суммой времён обработки множества последних задач, мы называем это: «установить текущий ПсевдоПакет». Когда J1 получен, P устанавливается в 0. После получения каждого последующего Ji, ПсевдоПакет сначала добавляет pi к P. Если P > 1, ПсевдоПакет пакетирует и устанавливает P в 0. Таким образом, i-тый псевдо-пакет содержит все элементы, кроме первого значения i-того пакета, вместе с первым значением (i+1)-ой партии, если только i = r. Каждая работа, кроме J1, принадлежит только одному псевдо-пакету.

Рисунок 4 показывает пример для p1 = 0.2, p2 = 0.6, p3 = 0.2, p4 = 0.3, p5 = 0.1, p6 – 1.1.

**Теорема 1** ПсевдоПакет имеет коэффициент приближения 2.

**Доказательство:** как и ранее, пусть Ci будет временем завершения оптимального графика для задачи . Пусть Ĉi обозначает время завершения заданий, когда алгоритм ПсевдоПакета запускается в экземпляре, и пусть mi будет количеством пакетов, созданных алгоритмом. Очевидно, у нас есть

и

Таким образом,

Благодаря Лемме 2 у нас есть

Пусть КаноническиЛучший будет алгоритм, который ставит задачу в каноническом порядке и время линейного алгоритма следующей секции (или алгоритма из [1]) для получения оптимального расписания пакетного списка при каноническом порядке. Очевидно, что мы имеем:

**Теорема 2** КаноническиЛучший имеет коэффициент приближения 2.

**Доказательство:** алгоритм ПсевдоПакета имеет коэффициент приближения 2 и является приоритетным алгоритмом. Учитывая пример задачи, КаноническиЛучший алгоритм выдает график со средним весом завершения не хуже, чем алгоритм ПсевдоПакета.

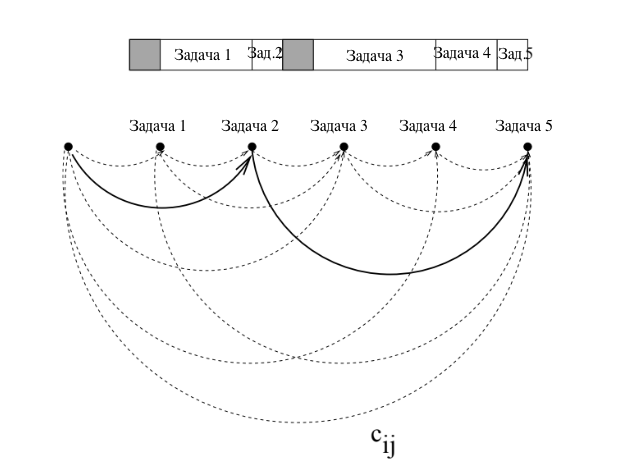


Рисунок 5 – Сведение задачи пакетной обработки к задаче пути.

**3 Альтернативный линейный алгоритм для пакетной обработки списка**

Теперь перейдем к времени выполнения КаноническиЛучшего алгоритма. Как упоминалось ранее, Брукер и Альберс [1] описали линейный алгоритм времени для задачи пакетной обработки списка. Здесь мы приведём простой альтернативный алгоритм. Алгоритм подобен их алгоритму, но в качестве подпрограммы используется хорошо известный алгоритм линейного поиска по матрице времени Лармора и Шибера [15].

Для этого предположим, что задачи 1,…,n и описаны в этом порядке. Затем, можно свести задачу пакетной рассылки к задаче кратчайшего пути следующим образом: построить взвешенный ориентированный ациклический граф G с узлами i=1,…,n (то есть один узел для каждого задания) и добавить фиктивный узел 0. Будет существовать ребро (i,j), если i<j. (см. схему на рисунке 5). Пусть ребро, стоящее ci,j для i < j, определено как

где s = 1 – время установки пакета. Кратко отметим:

**Лемма 3** Матрица C = (ci,j), определенные в (1), представляет собой матрицу Монжа для всех вариантов pi, wi ≥ 0. Более того, значения могут быть запрошены за О(1) время после линейной предварительной обработки.

**Доказательство:** пусть являются частичной суммой значений pi b wi. Тогда у нас есть

Для i < i’ и j < j’

Возвращаясь, теперь к обсуждению сокращения, легко увидеть (см. [1] для деталей), что стоимость пути < 0, i1, i2,…,ik, n > описывает значение расписания, которое устанавливается для каждого задания i1, i2,…,ik, n. И наоборот, любая дозировка со стоимостью А соответствует пути G с длиной A. Кратчайший путь может быть вычислен за время O (n2) с использованием следующей динамической программы:

Допустим

тогда

что приводит к таблице, в которой элементы могут быть вычислены строка за строкой (см. рисунок 6).

Другими словами, динамическая программа вычисляет минимумы строк матрицы E, где

с l = 2,…,n и k = 0,…,n – 1

**Лемма 4** Матрица E = (El,k) определена в (3), как матрица Монжа.



Рисунок 6 – Таблица динамического программирования.

**Доказательство:** матрица Монжа фиксируется при сложении и берёт минимум. Как оказалось, все минимумы строк могут быть вычислены в линейное время с использованием алгоритма Лармора и Шибера, который также известен, как алгоритм Ларша [15]. Для выполнения алгоритма Ларша, нам нужно лишь, чтобы матрица E удовлетворяла следующим условиям:

1. Для каждого индекса строки l в E существует индекс столбца γl такой, что для k>γl, El,k = ∞. Кроме того, γl ≤ γl+1.
2. Если k ≤ γl, тогда E[l,k] можно оценить за О(1) раз при условии, что минимумы первых l строк уже известны.
3. E – полностью монотонная матрица

Если эти условия выполняются, алгоритм Ларша рассчитывает минимумы каждый строки E во времени O(n). (См. также [5]).

Условие 1 понятно из того факта, что в матрице E все бесконечности находятся в верхнем треугольнике матрицы.

Перейдём к условию 2. Условие 2 описывает онлайн-протокол, лежащий в основе вычисления таблицы динамического программирования. В нём говорится, что элемент в столбце k «узнаваем», как только обнаружен минимум строки в строке k. Рисунок 7 это иллюстрирует. Например, если известен минимум строки 4 (то есть значение для E [4]), то становятся доступными значения столбца 4, поскольку эти значения имеют вид E[4] + c[.,.]. Кроме того эти значения могут быть запрошены за время О(1) из-за леммы 3.

Условие 3 следует из леммы 4.

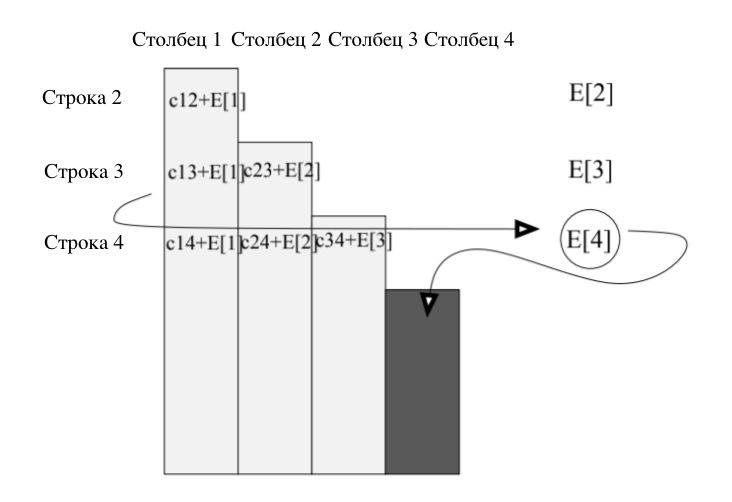


Рисунок 7 - «Онлайн» протокол таблицы

Поскольку КаноническиЛучший требует, чтобы приоритеты были отсортированы в первую очередь, в итоге мы имеем:

**Теорема 3** Алгоритм КаноническиЛучший имеет время выполнения O(n logn).

**4 Нижняя граница для приоритетных алгоритмов**

Теперь мы опишем нижнюю границу по коэффициенту аппроксимации любого приоритетного алгоритма. Чтобы показать нижнюю границу рассмотрим задачу, которая состоит ровно из двух заданий. Возможны два порядка задач. У каждого порядка есть два способа решения задачи; либо один пакет, состоящий из обоих частей, либо два отдельных пакета, каждый из которых содержит одну часть. Поскольку порядок задач в пакете не имеет значения, два решения идентичны и, таким образом, существует всего три возможности для Оптимума и КаноническиЛучшего.

Теперь, учитывая задачу {(p1, w1), (p2, w2)} мы предполагаем равные приоритеты , можно показать, что {(p1, w1), (p2, w2)} = {(1, 1 + ∈), (1 + , 1 + )} максимизирует C = . Отметим значение ∈, которое используется для разрыва связи и заставляет КаноническиЛучший выбрать порядок, отличный от Оптимум; однако ∈ будет стремиться к 0.

Мы используем квадратные скобки [ ] для обозначения пакета. Теперь рассмотрим:

Оптимум выбирает порядок (6), так как он даёт минимально возможную цену. КаноническиЛучший вынужден выбирать между (4) или (5) из-за небольшого увеличения приоритетов (1, 1 + , вызванного , перед началом этой задачи. Оба варианта имеют одинаковую стоимость. Таким образом:

Поскольку

**5 Эксперименты с адаптивным алгоритмом**

В предыдущем разделе мы продемонстрировали решение, которое даёт худший коэффициент аппроксимации для КаноническиЛучшего при наличии двух заданий. Остаётся вопрос, существует ли более сложная задача, учитывающая большее количество задач. Мы предполагаем, что нет. В этом разделе мы дадим результаты компьютерных экспериментов, которые подтверждают эту гипотезу.

Для исследования проблемных зон, состоящих из более, чем двух задач, был разработан эволюционный алгоритм. Индивидуумы с произвольным числом задач i, где 2 ≤ i ≤ 6. Число задач ограничено шестью, поскольку оптимальное решение было получено путём исчерпывающего поиска. Для индивидуума число задач и данных о работе представляют генетическую структуру этого индивидуума. Функция пригодности f(x), используемая для оценки пригодности отдельного x для включения в последовательные эволюции, является просто КаноническиЛучшим конкурентным соотношением для индивидуума.

Алгоритм рассажен одним индивидуумом, состоящем хотя бы из одной задачи. Любой индивидуум с одной задачей всегда будет иметь f(x) = 1.0; мутация используется для быстрого получения индивидуумов с двумя или более генами с f(x) > 1.0. Эволюционная среда поддерживает в общей сложности μ = 50 родителей. Используя просто й детерминированный процесс отбора, все пары родителей были спарены, чтобы произвести λ = μ2 = 2500 потомков. Из полученной популяции μ + λ = 2550, самые сильные 50 выжили для следующего поколения.

В эволюционных алгоритмах, потомство обычно является продуктом мутации и скрещивания своих родителей; однако в нашем поиске мы полагались только на мутации. Мутации родителя состояли из обработки задач (генов) родителя; любая отдельная задача имела вероятность возникновения мутации φmut = 0.1; в противном случае ребёнок просто получал копию. Было применено несколько типов мутаций. Мутации влияли либо на вес, либо на время обработки задачи.

* С вероятностью 0.25 значения можно масштабировать на случайную величину с целью введения случайной мутации, тем самым приземляясь на новые окрестности ландшафта.
* Значения могут удвоиться (вероятность 0.25) или значения могут быть уменьшены вдвое (вероятность 0.25), чтобы быстро сходиться в тех случаях, которые основаны на малых или больших значениях определенной задачи по отношению к другим.
* С вероятностью 0.25 значения могут также измениться на небольшие суммы в случае, если работа может быть затруднена из-за очень небольших изменений в текущем дизайне. (Это немного похоже на использование в разделе 4).

Таблица 1 – Улучшения во время случайного случая.

|  |  |  |
| --- | --- | --- |
| Поколение | f(x) | Задача |
| 49 | 1.109481 | {(1.01, 0.40), (3.20, 1.26)} |
| 144 | 1.109964 | {(1.01, 0.40), (3.20, 1.26)} |
| 284 | 1.111070 | {(0.99, 0.39), (3.20, 1.26)} |
| 1969 | 1.111199 | {(0.99, 0.39), (3.20, 1.26)} |
| 5705 | 1.111338 | {(0.99, 0.39), (3.20, 1.26)} |
| 9283 | 1.111842 | {(1.00, 0.40), (3.20, 1.26)} |
| 10487 | 1.111890 | {(1.00, 0.40), (3.20, 1.26)} |
| 10971 | 1.111920 | {(1.00, 0.40), (3.20, 1.26)} |
| 17910 | 1.111956 | {(1.00, 0.40), (3.20, 1.26)} |
| 26522 | 1.111957 | {(1.00, 0.40), (3.20, 1.26)} |

Наконец, с вероятностью 0.1 мы применили дополнительное изменение к ребёнку, где количество задач было изменено с вероятностью 0.5 для увеличения и с вероятностью 0.5 для уменьшения на одну задачу. Уменьшения были выполнены путём удаления случайной задачи с равной вероятностью. Добавление задачи было введено путём добавления случайной задачи, либо путём дублирования одной из уже существующих задач с равной вероятностью. Мы избегали изменений, если они приводили к размерам задач вне допустимого диапазона.

Когда алгоритм рассажен с {(1.0, 1.0)}, результирующая эволюция была неинтересной. В пятом поколении индивидуум был из {(1.00, 1.00), (4.00, 4.00)}, где f(x) = 1.1111. С этого времени вплоть до, по крайне мере, поколения 18813, μ состояло из 50 клонов этого индивидуума и, по-видимому, оставался стабильным, несмотря на существование известной, более сложной задачи (раздел 4), аналогичным образом, при рассаде с этой проблемой {(3.45, 3.45), (1.00, 1.00)}, μ сразу сходятся на 50 клонах этого индивидуума, без каких-либо изменений, вплоть до, по крайней мере, поколения 20553. Никаких других, более сложных задач не найдено.

Когда было рассажено со случайной задачей {(0.33, 0.45)} всё было немного интереснее. Ранние поколения показали более разнообразные μ, состоящие из шести уникальных индивидуумов; все с f(x) = 1.0. В поколении 49 был найден индивидуум {(1.01, 0.40), (3.20, 1,26)} с f(x) = 1.109481 и μ состоял из 50 клонов. Последующие поколения показали дальнейшие небольшие улучшения. Выборка этой эволюции показана в таблице 1.

Таблица 2 – Пример порядкового перехода

|  |  |
| --- | --- |
| Родитель 1  Родитель 2 | 184 637 |25 случайный слой 637  352718 |64 добавления подчеркнуты |
| Ребёнок | 218637 |45 ребёнок – допустимая перестановка |

Все улучшения, как оказалось, являются результатом небольших изменений мутаций, других мутаций, не способных произвести конкурентоспособных индивидуумов. По состоянию на поколение 72233 дальнейшее улучшение не обнаружено.

Теперь мы обратимся к несвязанной реализации под GAlib, объектно-ориентированной библиотекой Мэтью Уолла [19], разработанной в MIT. Потому что можно сделать оптимальный график по линейному времени, если порядок фиксирован (как описано в разделе 3) в , естественно рассмотреть генетический алгоритм, где зона поиска – это множество перестановок. Затем можно оценить пригодность каждого индивидуума – его средний вес завершения в линейном времени.

Мы должны определить мутацию и скрещивание. Мутация просто меняет два произвольных элемента перестановки. Для скрещивания важно разработать механизм, который бы сохранял некоторые оригинальные черты двух индивидуумов таким осмысленным образом, который привёл бы к двум новым перестановкам. В упорядоченном скрещивании, впервые использованном Принсоном (см. [17]), каждый берёт случайную последовательность перестановок первого родителя и вставляет её непосредственно в ребёнка. Как описано в таблице 2, ребёнок будет готов, принимая материал из скрещивания второго родителя, где элементы вставляются в дочерний элемент в том порядке, в котором они встречаются в этом родительском элементе, начиная с позиции второго выреза и игнорирую элементы, уже вставленные из первого родительского элемента.

Все эксперименты имели следующие общие параметры:

* Численность населения: 1000;
* Количество поколений: 5000;
* Количество задач: 100;
* Вероятность скрещивания: 0.85;
* Вероятность мутации: 0.005;

Результаты сравнивались с (консервативной) нижней границей леммы 2, и наши результаты последовательно давали решения в соотношении ρ = 1.59. Подробные результаты этих экспериментов приведены в [18].

**6 Выводы**

Мы описали алгоритм приближения приоритетов для задачи . Мы также показали, что ни один приоритетный алгоритм не может иметь коэффициент аппроксимации менее . Как мы уже отмечали, мы предполагаем, что алгоритм КаноническиЛучший соответствует этой нижней границе. Тем не менее, это интересная задача открытого исследования, доказать правильность этой гипотезы.

Обратите внимание, что нижняя граница выполняется только для приоритетных алгоритмов. Это актуальное исследование, чтобы понять, есть ли нижняя граница, даже если это предположение отбрасывается, или даёт схему полиномиального приближения в случае, если такой нижней границы не существует.

Отметим, что версия алгоритма ПсевдоПакет полезна для задачи с онлайн-пакетированием, и мы описали результаты для онлайн-случая в [4].

**Благодарность**

Вольфганг Бейн провёл это исследование во время посещения Техасского университета в Далласе во время творческого отпуска из университета Невады в Лас-Вегасе. Отпускная помощь от UNLV признательна. Авторы благодарят Чарльза Шилдса из Техасского университета в Далласе за ценные комментарии и предложения.

**Библиографический список**

[1] S. Albers and P. Brucker. The complexity of one-machine batching problems. Discrete Applied Mathematics, 47:87–10, 1993.

[2] P. Baptiste. Batching identical jobs. Mathematical Methods of Operation Research, 52:355–367, 2000.

[3] P. Baptiste and A. Jouglet. On minimizing total tardiness in a serialbatching problem. Operations Research, 35:107–115, 2001.

[4] W. Bein, L. Epstein, L. Larmore, and J. Noga. Optimally competitive list batching. In Algorithm Theory - SWAT 2004, volume 3111 of

[5] W. Bein, M. Golin, L. Larmore, and Y. Zhang. The Knuth-Yaoquadrangle-inequality speedup is a consequence of total-monotonicity. Transactions on Algorithms. to appear.

[6] P. Brucker. Scheduling Algorithms. Springer Verlag, 2004.

[7] P. Brucker, A. Gladky, H. Hoogeveen, M. Kovalyov, C. Potts, T. Tautenhahn, and S. van de Velde. Scheduling a batch processing machine. Journal of Scheduling, 1(1):31–54, 1998.

[8] P. Brucker and J. Hurink. Solving a chemical batch scheduling problem by local search. Annals of Operations Research, 96:17–38, 2000.

[9] P. Brucker, M. Y. Kovalyov, Y. M. Shafransky, and F. Werner. Batch scheduling with deadline on parallel machines. Annals of Operations. Research, 83:23–40, 1998.

[10] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on demand video server with batching. In Proceedings of the second ACM international conference on Multimedia, pages 15–23. ACM, 1994.

[11] D. R. Dooly, S. A. Goldman, and S. D. Scott. On-line analysis of the TCP acknowledgment delay problem. Journal of the ACM, 48(2):243–273, 2001.

[12] J. A. Hoogeveen and A. P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In Proc. 5th Conf. Integer Programming and Combinatorial Optimization (IPCO), pages 404–414, 1996.

[13] A. R. Karlin, C. Kenyon, and D. Randall. Dynamic TCP acknowledgment and other stories about e/(e-1). Algorithmica, 36(3):209–224, 2003.

[14] R. Kuik, M. Salomon, and L. N. van Wassenhove. Batching decisions:structure and models. European journal of operational research, 75:243–263, 1994.

[15] L. Larmore and B. Schieber. On-line dynamic programming with applications to the prediction of rna secondary structure. Journal of Algorithms, 12:490–515, 1991.

[16] C. N. Potts and L. N. Van Wassenhove. Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity. The Journal of the Operational Research Society, 43:395–406, 1992.

[17] C. Prins. Competitive genetic algorithms for the open-shop scheduling problem. Technical report, Ecole des Mines de Nantes, 1999.

[18] L. Raymond. Heuristics for batching jobs under weighted average completion time. Master’s Thesis, University of Nevada, Las Vegas, 2006.

[19] M. Wall. GAlib: A C++ Library of Genetic Algorithm Components. Cambridge, Massachusetts, version 2.4 edition, 1996. http://lancet.mit.edu/ga/.

[20] G. Zhang, X. Cai, C. Y. Lee, and F. Wong. Minimizing makespan on a single batch processing machine with nonidentical job sizes. Naval Research Logistics, 48:226–240, 2001.