# Оптимальное пакетное расписание для параллельных машин

**Аннотация**. Мы рассматриваем проблему пакетного планирования на параллельных машинах, где задания имеют время выпуска, сроки и идентичное время обработки. Цель состоит в том, чтобы запланировать эти задания партиями размером не более *B* на *m* идентичных машинах. Предыдущая работа по этой проблеме была в основном сосредоточена на поиске возможных расписаний. Основываясь на проблеме минимизации энергии, мы рассматриваем проблемы, когда количество партий является значительным. Для минимизации количества пакетов на одном процессоре ранее требовался непрактичный алгоритм динамического программирования O(). Мы представляем алгоритм O() для одновременной минимизации количества партий и максимального времени завершения и даем улучшенные гарантии для вариантов с бесконечными партиями, приемлемым временем выпуска и партиями «бюджетами». Наконец, мы даем псевдо-полиномиальный алгоритм для общих целевых функций, чувствительных к количеству партий, и исправление ошибок в предыдущих результатах.

**Ключевые слова:** планирование, пакетирование, оптимальные алгоритмы.

## 1. Введение

Пакетное планирование относится к планированию заданий, когда задания могут обрабатываться партиями размером не более *B*. Концепция параллельного пакетного планирования заданий первоначально была предложена для моделирования поставок на грузовиках ограниченной вместимости [9]. Помимо прочего, она использовалась для моделирования управления крупными системами мультимедиа по запросу [2] и операциями «прожигания» в печи, где одновременно можно выпекать несколько чипов [10]. Мы ориентируемся на версию, в которой все задания в пакете обрабатываются вместе и запускаются одновременно. Кроме того, для каждого задания () расписание должно учитывать время выпуска () и сроки (), время, когда задания становятся доступными для обработки и должны обрабатываться соответственно. Это может, например, моделировать доставку людей, летящих в аэропорт для конференции, где каждый человек должен быть доставлен к указанному сроку с использованием парка транспортных средств с ограниченной вместимостью.

Многие результаты приводят к тому, что детерминированное планирование пакетов фокусируется на версии, где все задания имеют время выпуска, сроки и одинаковую длину *p* [5, 10, 9, 3, 1], где цель состоит в том, чтобы найти выполнимое расписание партий, каждая из которых содержит в большинство рабочих мест *B*. Время начала и окончания пакета должно соответствовать времени выпуска и крайнему сроку каждого задания в пакете. Используя стандартные методы, эти алгоритмы осуществимости могут быть использованы для минимизации таких целей, как максимальное запаздывание () и максимальное время завершения (). Обратите внимание, что, когда задания имеют различную длину, решение о выполнимости становится проблемой NP-полной, хотя существуют алгоритмы аппроксимации, например, [2].

Мотивированные вопросами экономии энергии, Chang et al. [4] недавно рассмотрели проблему минимизации времени использования машины, называемую временем активации. В случае одинаковой длины заданий это означает планирование всех заданий с использованием наименьшего количества пакетов. Чанг и соавт. [4] разработали алгоритм O() для этой задачи на основе работы Баптиста [1]; сложность пространства также очень высока, и она также предназначена только для случая с одним процессором. В статье также рассматривается множество других случаев проблемы активации, например, когда время выпуска и сроки являются интегральными и *р* = 1, они представляют собой линейный алгоритм времени. Несмотря на то, что планирование пакетов изучалось более двадцати лет, это первые алгоритмы, которые явно направлены на минимизацию количества пакетов. Однако мы ожидаем, что количество используемых партий почти всегда влияет на стоимость энергии (и, следовательно, на производительность) системы.

Основная проблема, которая решается, заключается в создании расписания партий для *m* идентичных машин. Партия (или экземпляр партии) *B* в расписании связана с тремя свойствами:

* Набор заданий, содержащихся в пакете. Обозначим через количество заданий в пакете. В выполнимом расписании , где *B* - данное ограничение размера партии.
* Время начала *s().* , время завершения = *s()*+ *p*. В выполнимом графике, + *p* dj.
* Машина *m(),* на которую запланирована партия, занимающая временной интервал [*s(*); *s(*+*p*)] на этой машине. В допустимом графике временные интервалы партий, запланированных на одной и той же машине, должны быть непересекающимися.

Наши результаты коротко: мы можем найти минимальное расписание подсчета партий и за время O() и улучшить его в вариантах задачи; мы создаем алгоритм O(n) для приемлемой задачи (). Мы даем псевдополиномиальный алгоритм для понятия целевых функций, чувствительных к количеству партий.

## 1.1 Связанная работа

Ikura и Gimple первоначально предложили алгоритм O() для планирования приемлемых заданий на одном процессоре с целью минимизации . Lee и др. Нашли алгоритм O(*nB*) для этой же проблемы, используя динамическое программирование [10]. Баптист [1] окончательно показал, что проблема с произвольными временами выпуска была разрешима за полиномиальное время для широкого класса целей функции суммы, таких как . Однако его алгоритмы имеют чрезвычайно высокую (полиномиальную) сложность.

Недавно Condotta et al. [5] разработали улучшенные алгоритмы для задачи выполнимости для общих сроков выпуска и сроков: для случая с одной машиной они обеспечивают алгоритм времени O(). Они также изучают ранее проигнорированный случай нескольких идентичных машин и предоставляют алгоритм времени O(). Эти алгоритмы являются обобщенными формами алгоритмов для задачи, не связанной с пакетированием (*B* = 1): алгоритм O() основан на методе «запрещенных областей» Гарея и др. [7], а O() алгоритм для многопроцессорного случая основан на методе «барьеров» Саймонса [14].

Методы «барьеры» и «запретные области» для одного процессора отличаются тем, что они выбирают расписания с тем свойством, что каждое задание, пронумерованное слева (или справа), начинается как можно скорее. Формально время начала i-го задания слева в сгенерированном расписании является нижней границей времени начала i-го задания в любом возможном расписании. Эти графики, таким образом, являются оптимальными для целей и . Скажем, что эти графики имеют оптимальную по модулю структуру. Современные алгоритмы используют теоретико-графические приемы и графики с одинаковой структурой [6, 12].

Condotta et al. в статье утверждают, что каждая партия, считая слева, для своих барьеров имеет минимальное время запуска (лемма 4). Это утверждение неверно: рассмотрим экземпляр проблемы с большими крайними сроками, двумя компьютерами, B = 2 (или любым четным числом), заданиями B, выпущенными в момент времени 0, и заданиями B, выпущенными в момент p. Алгоритм барьеров создаст график с двумя полными партиями, второй в момент времени p. Однако существует выполнимое расписание, когда первые две партии запускаются в нулевое время, каждая из которых содержит B/2 задания. Это опровергает их утверждение и лишает законной силы их доказательство правильности. Однако, исправляя это утверждение, все еще можно показать правильность их алгоритма. (Вот набросок: рассмотрим только те классы расписаний, где каждый пакет слева жадно выполняет столько заданий, сколько возможно. Расписания, генерируемые алгоритмом барьеров, - это те, которые для любого k, оба обрабатывают минимальное количество заданий в первые k партий пронумерованных слева и запускают каждую партию не раньше, чем любой непустой в любом другом расписании этого класса. Обработка минимального количества заданий имеет решающее значение для доказательства того, что время запуска партии сведено к минимуму.)

Они также утверждают, что их алгоритмы немедленно минимизируют и в задаче пакетирования. Мы не считаем, что это так. Рассмотрим случай с одной машиной: при незначительной задержке задания может оказаться возможным перекрыть его в пакете с другими заданиями, что значительно сократит время его завершения, не блокируя обработку первого задания. Алгоритм барьеров создает барьеры только тогда, когда он сталкивается с невозможностью выполнения, поэтому, если он никогда не сталкивается с невозможностью выполнения, не предпринимается никаких попыток отложить задания для пакетирования их вместе с позднее выпущенными заданиями. Точно так же алгоритм запрещенных областей не найдет запрещенных областей.

Следующий простой пример продемонстрирует наши претензии. Запустите алгоритм барьеров для заданий с помощью ; пары {(1, 16), (2, 20), (6, 24)) }, с длиной обработки для заданий p = 8, с размером партии B = 3 и одним станком: партия будет создана в моменты времени = 1 и при + p = 9. Оптимальное расписание для использует только одну партию, начиная с = 6. Интересно, что оптимальное расписание для использует одну партию, начиная с = 2, а другую - при + p = 10.

**Теорема 1**. В задаче о пакетировании существуют случаи, когда минимизация и одновременно невозможна.

С другой стороны, при планировании заданий на нескольких процессорах Саймонс [14] показал, что w.l.o.g. можно рассматривать только циклические графики. Мы сделаем именно это предположение в нашей статье. Оригинальное доказательство следующего утверждения исходит от Саймонса для случая не пакетирования [14].

**Лемма 1**. Для любого возможного расписания существует идентичное решение, кроме машинного назначения, которое является циклическим, то есть где запланированы на той же машине.

## 1.2. Наш подход

Обобщим понятие единицы оптимальности. Мы будем называть нашу структуру право-тяжелой пакетно-оптимальной (right-heavy batch-optimality, RHBO). Она включает в себя следующие свойства (обратите внимание на схему нисходящей пакетной нумерации):

1. Рассмотрим любое возможное расписание , состоящее из партий , где - самая ранняя стартовая партия (а - самая последняя), содержащая задание в расписании . ; то есть количество рабочих мест в является верхней границей для возможных графиков.
2. Для любого время начала партии является нижней границей для возможных расписаний; то есть для любого в любом возможном расписании , .

В случае, когда B = 1, первое свойство тривиально, а второе свойство делает структуру идентичной единичной оптимальности. Обратите внимание, что в отличие от исправленной версии алгоритма барьеров, наши оценки верны для всех возможных расписаний. Любой график с этими свойствами является оптимальным для многих целей:

1. , потому что время начала является нижней границей. Фактически, рабочий диапазон (время доступности) = всех машин сведен к минимуму; так, например (средний рабочий диапазон) и ряд других норм также сведены к минимуму.
2. *K*, количество партий, по первому свойству.
3. , сумма времен запуска партии, поскольку используется минимальное количество партий, а время начала каждой партии является нижней границей.

В первом разделе нашей статьи приведен алгоритм с низкой полиномиальной временной сложностью, свидетельствующий о существовании этих структур. Мы также используем этот результат существования для получения оптимального рекурсивного алгоритма для приемлемой задачи планирования партии [9]. Свойство одновременной минимизации рабочего времени на нескольких машинах имеет решающее значение для разложения.

В общем, алгоритмы Condotta et al. [5] будет эффективно использовать пакеты только в том случае, если эта часть расписания сильно ограничена или многие задания совместно используют время выпуска. Когда размеры партии больше, чем, например, B = 2, это становится очевидным. Используя меньше пакетов, мы также можем улучшить нашу временную сложность, связанную с тем, что существует допустимое расписание с *K\** партиями (*n/B ≤ K\*≤ n*), поскольку избыточные партии увеличивают алгоритмические издержки. В случае приемлемого времени выпуска мы создаем элегантный алгоритм, который ищет rhbo-расписания. Это более общая и более низкая сложность, чем предыдущие алгоритмы для этой задачи. Это завершает наше изучение графиков rhbo.

Наконец, мы разрабатываем псевдо-полиномиальный алгоритм для оптимизации широкого спектра задач, чувствительных к количеству партий, обобщая [1]. Отсутствие структуры в этом общем случае приводит к очень высокой сложности. Этот результат, доказательства вспомогательных лемм и псевдо-кодовых версий алгоритмов опущены по пространственным причинам; полная версия находится по адресу <http://www.cs.umd.edu/~samir/grant/BatchScheduling.pdf/>

## 2. Планирование заданий на нескольких пакетных машинах

В этом разделе мы изучим проблему планирования всех заданий в данном случае. Таким образом, когда мы ссылаемся на выполнимое расписание, оно должно успешно обрабатывать все n заданий. Мы будем работать через ряд предварительных (неосуществимых) графиков в наших алгоритмах. Каждое предварительное расписание будет подчиняться структуре rhbo: мы называем два свойства расписания rhbo как Инвариант (1) и Инвариант (2), соответствующие нумерации в определении. Мы говорим, что задание доступно в срок в партии , если . Используя это понятие, мы определим третий инвариант, который определяет выбор работы в пакетном режиме:

(3) таким образом, что x <y, если Jj доступен в Bx до крайнего срока, то Bx полон заданий с не менее строгим временем выпуска ().

Этот инвариант можно рассматривать непосредственно как выражение связи между отдельным пакетом Bx и набором предыдущих заданий в пакетах с более высоким (более ранним) номером пакетоа. В нем также говорится, что каждый пакет должен предпочитать выбирать последние выпущенные задания из набора заданий, предшествующих следующему самому раннему пакету. Обратите внимание, что увеличение времени запуска может только уменьшить набор доступных для крайнего срока заданий и, таким образом, только облегчает выполнение этого инварианта.

Мы предполагаем, что w.l.o.g. : задания, нарушающие это ограничение, невозможно обработать. Первоначально пусть для всех Bb (включая те, которые раньше, чем Bn, которые фактически не могут быть использованы), поскольку является тривиальной нижней границей времени начала любого пакета.

**Лемма 2.** Инварианты (2) и (3) влекут инвариант (1).

Доказательство. Предположим, что инвариант (1) нарушается, тогда как два других инварианта выполняются. Пусть будет последней партией в выполнимом графике , такой, что . Поскольку мы выбрали последнюю партию, в которой нарушен инвариант, инвариант сохраняется для , и поэтому должен содержать хотя бы одно дополнительное задание , которого нет в . Поскольку . не может быть полным из инварианта (3) следует, что . По инварианту (2) и, следовательно, . Крайний срок для работы нарушен, поэтому график не может быть осуществимым.

**Лемма 3**. Если инварианты оптимальности выполняются для частичного расписания , то для любого возможного расписания , состоящего из должно быть верно, что .

Доказательство. Пусть . Если , то имеем .. В противном случае (): поскольку и , по инварианту (1) существует такое задание , что . Также по инварианту (2) , поэтому по инварианту (3) . Поскольку с , как и выше .

## 2.1 Планирование с неограниченным числом машин

**Теорема 2**. Возможное расписание, подчиняющееся инвариантам оптимальности, может быть вычислено за время O (), если m = ∞.

Доказательство. Для первой (последней) партии является нижней границей времени начала - таким образом, установка = подчиняется инварианту (2). Инвариант (3) определяет, что этот пакет должен быть заполнен максимальным числом (до B) из последних выпущенных крайних сроков работ. Для любой другой партии можно индуктивно предположить, что частичное расписание подчиняется инвариантам. Пусть U будет множество незапланированных заданий. Все задания в U могут быть запланированы только в и более ранних партиях. Установим , чтобы оно было последним временем выпуска в U; Лемма 3 гарантирует, что это удовлетворяет инварианту (2). Еще раз, инвариант (3) диктует, что максимальное число последних выпусков, доступных до крайнего срока, выбирается для каждой партии.

Каждая созданная партия содержит хотя бы одну работу. Таким образом, существует не более n пакетов, и эта конструкция занимает O () времени.

Примерное расписание показано на рисунке 1, на основе входных данных из таблицы 1 с партиями, занимающими три единицы времени (p = 3) для обработки до двух (B = 2) заданий одновременно.

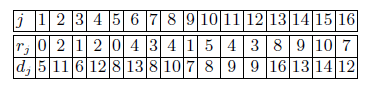


Таблица 1. Задания для примера 2

## 2.2 Планирование с ограниченным числом машин

**Теорема 3**. Учитывая предварительное расписание, содержащее все задания с не более чем B заданиями в любом пакете, и соблюдая инварианты оптимальности, за время O() можно либо показать, что не существует допустимого расписания, либо найти возможное расписание, подчиняющееся инварианты оптимальности.

Доказательство. Мы покажем, как использовать сохраняющие инварианты преобразования, чтобы сделать этот график выполнимым. Мы используем два совместных чередующихся прохода: PushForward, который увеличивает время запуска, и MoveBack, который перемещает задания, которые предположительно находятся в неправильном пакете, назад.

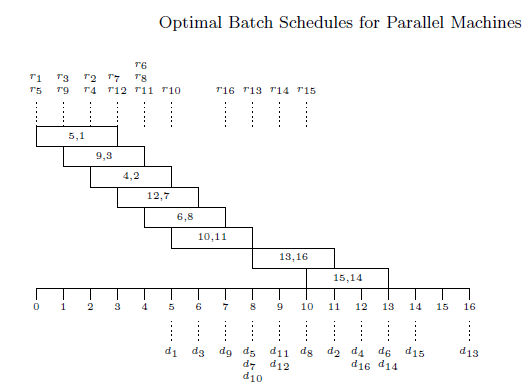


Рис. 1. График, построенный по теореме 2 на примере 2

PushForward - это проход, который начинается первым. Он обрабатывает партии слева направо (от самого раннего к последнему) последовательно; изначально он начинается с , самой ранней непустой партии в расписании. Опишем его действие. Пусть будет текущей партией. Пусть P будет набором партий, которые являются более ранними, чем (с более высоким индексированием): индуктивно предположим, что пакеты в P (1) находятся в неубывающем порядке времени начала (считая сначала более индексированные партии), (2) можно запланировать без перекрытия на m машинах и (3) содержат задания, время выпуска которых и (4) сроки соблюдены. Для обработки , пусть



Как отмечалось ранее, увеличение времени запуска всегда сохраняет инвариант (3); теперь мы покажем, что инвариант (2) также поддерживается. Для любого из первых трех терминов возможно, что они могут ссылаться на пустую партию после окончания расписания: для всех таких партий их время начала установлено равным -∞, поэтому термин сокращается до -∞, что является правильной нижней границей. В противном случае (нормальный случай) первый член является действительной нижней границей, потому что мы ограничились классом циклических расписаний, и не может быть перекрытия между партиями, запущенными на одной машине. Это также удовлетворяет индуктивной гипотезе (2). Для третьего члена по определению нижняя граница для времени начала распространяется на . Это удовлетворяет индуктивной гипотезе (1). Окончательное множество времен освобождения является действительной нижней границей по лемме 3, удовлетворяющей индуктивной гипотезе (3).

После обновления времени запуска, если в есть какие-либо задания, которые более недоступны по срокам, выберите один такой произвольно и переходите к следующему этапу MoveBack. Если таких работ нет, то основная гипотеза (4) выполняется. Если = , то завершить: предположим, что все партии в нашем расписании подчиняются ограничению размера партии (которое мы еще не показали), тогда при использовании наших индуктивных гипотез требования для допустимого графика удовлетворяются. В противном случае () переходите к следующей партии ().

Теперь мы опишем MoveBack. Эта фаза не будет регулировать время запуска, поэтому инвариант (2) сохраняется. Мы будем изучать инвариант (3) отдельно для каждого пакета и его набора предыдущих заданий, чтобы показать, что он выполняется для всех пакетов (когда это очевидно, мы оставим неявным, какой пакет инвариант сохраняется относительно). Опишем действие этого этапа. Первое действие, которое предпринимает эта фаза - удалить из . Если не существует предшествующих заданным по сроку заданий для , это не влияет на инвариант (3) относительно . Если существует хотя бы одно такое задание, выберите задание с последним временем выпуска и переместите его из текущей партии в . В этом случае мы говорим, что работа была перенесена из . Это может нарушить инвариант (3) по отношению к ; если это так, мы покажем, что инвариант восстанавливается до конца этого этапа. Удаление задания гарантирует, что . Остальная часть этой фазы перемещается справа налево в последовательных пакетах, начиная с . Вызов текущей обрабатываемой партии ; также позвольте текущей работе, первоначально , называться . Теперь опишем действие, выполненное для ; помните, что когда мы говорим, что эта фаза продолжается, это означает, что следующая проверенная партия - это предыдущая партия .

**Случай 1** (. Пусть .

**Случай 1.a** (. Поменяйте местами в , удалив . Продолжить MoveBack с .

**Случай 1.b** (). Продолжить MoveBack с .

В любом случае, новое, возможно, доступное по срокам, задание теперь будет предшествовать ( или ). Даже если задание доступно в срок, его время выпуска не превышает наименьшее в , поэтому инвариант (3) сохраняется.

**Случай 2** (). Поместите в .

**Случай 2.a (Работа была перенесена из )** Предположим, . Поскольку предшествовал перед его перемещением (после выполнения предыдущего этапа) и , по инварианту (3) не мог быть доступен в до крайнего срока. Тем не менее, он доступен для крайнего срока в , и благодаря действию PushForward мы знаем, что это означает, что доступен для крайнего срока во всех более ранних (с более высоким номером) пакетах. Противоречие . Поскольку поступил из некоторой партии позже, чем , но не позже, чем , и был доступен в крайнем сроке в этой исходной партии, по инварианту (3). Поэтому замена на не может нарушать инвариант (3) относительно .

**Случай 2.b (Задание не было перенесено)**. Добавление дополнительного задания в неполную партию не может нарушить инвариант (3), поэтому оно сохраняется.

В любом случае преобразования этой фазы завершены. Только партии между и включительно были изменены. Что касается пакетов с *f> a* для *f <c*, это означает, что инвариант (3) был сохранен. Таким образом, мы показали, что для каждой партии инвариант (3) выполняется по отношению к нему в конце этой фазы. Напомним, что теперь инвариант (1) выполняется по лемме 2. Если *z > n*, объявить экземпляр планирования недопустимым: по инварианту (1) в может быть запланировано только n - 1 заданий. В противном случае перейдите к PushForward в : поскольку мы ранее не изменили какие-либо пакеты , для них сохраняются необходимые индуктивные гипотезы.

Это завершает описание самого алгоритма. Как отмечалось ранее, мы все еще должны показать, что ограничение размера пакета выполняется, чтобы показать, что алгоритм частично корректен: если он завершается, он дает правильный ответ. Напомним, мы требовали, чтобы наше первоначальное расписание подчинялось ограничению. Только передача MoveBack изменяет назначение заданий пакетам, но добавляет только чистое задание к пакету, который имеет не более B -1 заданий. Поэтому ограничение размера партии всегда соблюдается.

Теперь мы должны показать, что наш алгоритм завершается. Мы утверждаем, что может быть не более O () проходов: для каждого прохода MoveBack никогда не может быть снова помещен в , потому что время начала только увеличивается, а задания переносятся только, если они доступны в срок; Есть *n* заданий и не более *n* пакетов, поэтому это делает возможным O () проходов. Оба прохода выполняются за время O(*n*), поэтому следует ограничение времени O ().

Полный алгоритм формируется путем составления двух предыдущих теорем: выполнимые расписания для конечного m являются подклассом расписаний для неограниченного m, поэтому предварительное условие для теоремы 3 выполнено. См. Рисунок 2 и рисунок 3, где m = 2. Однако, поскольку теорема 3 мало требует от своего первоначального расписания, гораздо менее разумные схемы дадут те же временные ограничения.

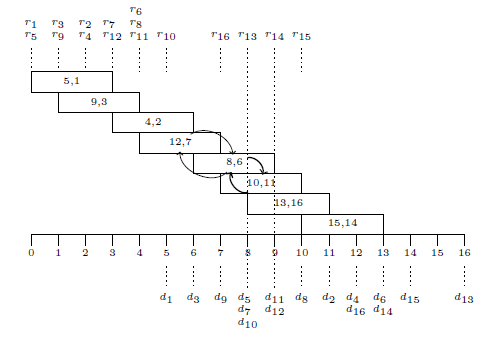


Рис. 2. Операции, выполняемые первым (толстые стрелки) и третьим (тонким) проходом MoveBack

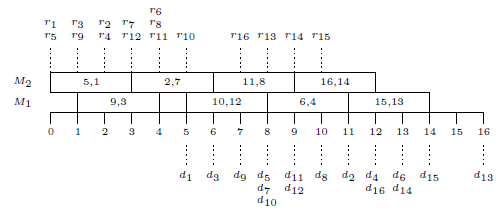


Рис. 3. Окончательный график, подготовленный для Примера 2

Мы еще не обсуждали, как эффективно представлять содержимое партии. Пусть содержимое каждого пакета представлено двумя структурами данных: двоичная минимальная куча заданий, упорядоченных по срокам, и avl-дерево времени выпуска заданий, поддерживающего счетчики в каждом узле для дублированных времен высвобождения. Наши доказательства эффективности опущены по космическим причинам; они немного изменяют внутреннюю часть алгоритма, чтобы улучшить его производительность. Если нам дан фиксированный бюджет партии K\* (модифицирующий алгоритм для выхода после превышения его бюджета партий, а не n партий), мы можем назвать этот бюджет K\*, и улучшенные границы сохранятся; альтернативно, если существует допустимое расписание с партиями K\*, эта граница также выполняется.

**Следствие 1**. Алгоритм пакетного бюджета равен

**Следствие 2**. Алгоритм завершается за время O () в течение приемлемого времени выпуска. Алгоритм пакетного бюджета - *O(nK\*).*

**Следствие 3**. Алгоритм завершается за O() времени для неограниченного случая (B = 1). Алгоритм пакетного бюджета - O ().

Используя расширенный подход двоичного поиска, изложенный в Condotta et al. [5] минимизация может быть выполнена с O() фазой предварительной обработки и вызовом алгоритма O() раз. Кроме того, следуя подходу, изложенному в Condotta et al. [5], мы можем легко соблюдать ограничения старта-старта. Во-первых, расписание может быть пройдено через этап предварительной обработки O (), что гарантирует, что если задание *a* предшествует заданию *b*, то и . После генерации расписания замена создаст расписание, которое подчиняется ограничениям приоритета.

## 3.Планирование согласованных заданий

Теперь мы разработали более быстрый алгоритм для решения проблемы с приемлемыми заданиями, где . Мы предполагаем, что задания сортируются по возрастанию срока (с учетом неуменьшающегося времени выпуска). Опишем структуру решения, которое мы ищем. По нашему предыдущему результату, если существует допустимое частичное расписание, существует частичное расписание rhbo. С помощью простого перестановочного аргумента [10], который не нарушает наши инварианты, мы также предполагаем, что w.l.o.g. и что каждая партия состоит из последовательно пронумерованных заданий. Наконец, отметим, что эти графики «смещены влево» (см., например, [1]). Это подразумевает, что при назначении заданий пакетам время начала пакета полностью определено: оно должно быть максимумом для всех и времени, когда предыдущая партия на машине завершает (назначения машины остаются определенными циклическим планированием).

Нам нужно будет поддерживать списки времени доступности машины: для этого мы используем чисто функциональные очереди [8, 13]. Нам даны три функции: head(Q) возвращает начало очереди Q, tail(Q) возвращает Q с удаленным фронтом, а snoc(X; Q) создает новую очередь с X, вставленным в конец Q. Все из этих операций O (1) и неразрушающие. Списки времени доступности будут поддерживаться в отсортированном порядке возрастания, так что заголовок (A) является самым ранним временем доступности в A. Мы определяем новую операцию, U(q; t) = snoc(tail (q); t): это будет использоваться для обновления времени доступности: когда запланирован новый пакет, заканчивающийся в момент времени t, с помощью циклического планирования он выполняется на той же машине, что и в результирующем расписании, ранее (в предыдущем частичном расписании) .

Теперь мы можем легко описать фактический алгоритм. Пусть будет определен (см. Ниже) так, что - это самая ранняя работа, которую можно выполнить вместе с в выполнимом графике. Рассмотрим выполнимый график rhbo для заданий *i*: согласно Invariant (1) последний пакет должен состоять из заданий ,…,. После удаления этой конечной партии обратите внимание на то, что для первых заданий оставлено допустимое расписание. Таким образом, мы индуктивно предполагаем, что у нас есть расписания rhbo для каждого из первых *j <i (i ≤ n)* заданий (из которых мы можем вычислить L), а затем найти единственное возможное расписание rhbo для заданий *i* (или не выполняется, если их нет) , Обратите внимание, что *L* - неубывающая функция (): это наблюдение делает табуляцию более эффективной. - это список времени доступности для расписания rhbo первых заданий. *E(j,i) = max{; head()} + p* - конечное время смещения влево последнего пакета в расписании для заданий *i*, где расписание состоит из пакета заданий ,…,, добавленного в расписание rhbo для первых заданий *j* , Формально:



Если в какой-то момент не определен, поскольку он минимизирует пустой набор, не может быть никакого rhbo-расписания и, следовательно, вообще никакого выполнимого расписания. *E* и *U* не приведены в таблице в динамической программе. и могут быть вычислены за O(n) время. В случае целочисленного времени выпуска и сроков, алгоритм двоичного поиска для , созданный Lee et al. [10] можно объединить с нашим алгоритмом для решения многопроцессорной задачи за O(.

## 3.1 Согласованное время обработки

Релаксация к согласовнным временам обработки была впервые изучена Ли и Ли [11]. Многопроцессорное планирование без времени выпуска и одного крайнего срока ( = *d*), которое обязательно согласуется со временем обработки, является унарным NP-Hard. Однако наш алгоритм легко адаптируется к случаю с одним процессором.



## 4. Выводы

Твердость многопроцессорного пакетного планирования для целей, не удовлетворяемых структурой rhbo, остается открытой проблемой: наилучшим ли является псевдо-полиномиальный алгоритм? Если да, то каковы наилучшие алгоритмы приближения? Большинство из этих проблем открыты, даже если B = 1; является заметным исключением. Из-за теоремы 1 может быть трудно эффективно минимизировать .