[INFORMS Journal on Computing](https://pubsonline.informs.org/journal/ijoc) [Vol. 21, No. 3](https://pubsonline.informs.org/toc/ijoc/21/3) P.398-410

**Одновременное пакетирование и планирование с использованием динамической декомпозиции на сетке**

Майкл С. Феррис

Отдел компьютерных наук, Университет Висконсина - Мэдисон

Уэст-Дейтон-стрит, 1210, Мэдисон, Висконсин, 53706, США, [ferris@cs.wisc.edu](mailto:ferris@cs.wisc.edu)

Арул Сундарамурти, Христос Т. Маравелиас

Кафедра химической и биологической инженерии, Висконсинский университет - Мэдисон

1415 инженер, доктор, Мэдисон, WI 53506, США {sundaramoort@wisc.edu, [maravelias@wisc.edu](mailto:maravelias@wisc.edu)}

Проблемы планирования возникают во многих приложениях в обрабатывающих отраслях. Тем не менее, несмотря на различные усилия в разработке эффективных методов планирования, современные подходы не могут быть использованы для разрешения случаев промышленного значения в разумные сроки. Целью данной статьи является разработка схемы динамического разложения, которая использует структуру проблемы и хорошо подходит для грид-вычислений. Проблема, которую мы изучаем, заключается в одновременном пакетировании и составлении расписания многоступенчатых пакетных процессов. Алгоритм сначала разбивает исходную проблему на подзадачи 1-го уровня на основе пакетных решений. Подзадачи, которые остаются нерешенными в пределах лимита ресурсов, затем динамически разлагаются на подзадачи 2-го уровня на основе решений о назначении единиц пакета за один этап. Процесс может быть повторен динамическим образом путем определения подзадач, которые не могут быть решены в рамках заданного ограничения ресурсов и их разложения путем назначения пакетных единиц, пока не будут решены все подзадачи. Альтернативно, проблема может быть разложена на ряд многообещающих подзадач с использованием автоматической схемы сильного ветвления. Наши результаты показывают, что предложенный метод может быть использован на сеточном компьютере для решения больших задач с оптимальностью в разумные вычислительные сроки.

Ключевые слова: химические периодические процессы; пакетирование и планирование; смешанно-целочисленное программирование; сеточные вычисления; алгоритм разложения.

1. **Введение**

Планирование производства является очень важной и сложной задачей во многих перерабатывающих отраслях. Например, в нефтяном секторе проблемы с расписанием возникают при транспортировке и выгрузке сырой нефти из трубопроводов, передаче в резервуары для хранения и загрузке для каждой единицы перегонки сырой нефти. В основных химических и потребительских товарах проблемы планирования обычно включают в себя последовательность производства в единицах непрерывной обработки, в то время как в фармацевтическом и специализированном химическом секторах проблемы планирования включают распределение ограниченных ресурсов (например, единиц обработки и хранения) конкурирующим продуктам и последовательность партий на юниты.

С точки зрения конфигурации установки, наиболее распространенной структурой периодических процессов является многостадийная многопродуктовая пакетная структура, где несколько продуктов конкурируют за набор ограниченных ресурсов, каждый продукт должен последовательно обрабатываться в несколько этапов, и каждый этап содержит один или несколько параллельных блоков обработки (см. рисунок 1). Порционное разделение и смешивание обычно не допускаются, и нет рециркуляционных потоков. Обратите внимание, что проблемы потока и нескольких блоков (или одноэтапных) можно рассматривать как частные случаи многоэтапной проблемы (Pinedo 2002). Кроме того, различные политики хранения, такие как ограниченное / неограниченное / отсутствие промежуточных политик хранения и передачи, могут применяться в зависимости от характера продуктов и доступности ресурсов.

Планирование многоступенчатых пакетных процессов привлекло значительное внимание исследователей из сообщества разработчиков систем процессов (см. обзоры Pinto и Grossmann, 1998; Pekny and Reklaitis, 1998; Shah, 1998; и Méndez и др., 2006). Существующие способы предполагают, что решения о пакетировании и планировании принимаются независимо, то есть каждый заказ делится на несколько пакетов (пакетирование), которые затем назначаются единицам обработки и упорядочиваются (планирование). Для секвенирования партий существуют два различных подхода: I) на основе слотов (Pinto и Grossmann 1996, Lamba and Karimi 2002) и II) на основе последовательностей (Hui and Gupta, 2000; Mendez и др., 2001; Gupta и Karimi, 2003). Для решения более крупных случаев несколько исследователей разработали методы декомпозиции. Harjunkoski and Grossmann (2002), Maravelias and Grossmann (2004) и Roe и др. (2005) разработали гибридные методы, в которых используются методы смешанного целочисленного программирования (MIP) и программирования с ограничениями (CP). Маравелиас (2006) представил технику декомпозиции, которая сочетает в себе MIP и алгоритмы для конкретных задач. Наконец, Neumann и др. (2002) и Kelly (2002, 2003) разработали алгоритмы разложения на основе MIP (обзор Mendez и др. (2006) для обзора наиболее широко используемых методов разложения). Наконец, Prasad и Maravelias (2007) разработали MIP-формулировку для одновременного дозирования и планирования многостадийных периодических процессов. Было показано, что их формулировка дает лучшие решения, чем предыдущие последовательные подходы, где пакетирование выполняется независимо от календарного планирования.

Хотя временной интервал планирования проблем в химической промышленности обычно варьируется от одной недели до одного месяца, технологическое оборудование может быть перепланировано на ежедневной основе. Это связано с тем, что новые заказы с жесткими сроками часто приходится вставлять в существующие графики, даты выпуска должны обновляться из-за задержек в поставках сырья, а задержки обработки и сбои оборудования вызывают серьезные сбои. Поэтому разработка методов решения, которые позволяют нам решать реальные проблемы за несколько часов, является обязательной. Однако, несмотря на достижения в области компьютерного оборудования и программного обеспечения для оптимизации, существующих подходов недостаточно для решения реальных проблем. Конкретное планирование, в частности, остается общеизвестно сложной проблемой.

Предметом данной статьи является разработка алгоритма динамической декомпозиции, который может использовать вычислительные ресурсы грид-вычислений и системы управления Condor, что позволяет нам решать большие задачи планирования. Наша цель состоит в том, чтобы показать, что, если знание конкретных проблем используется для разработки метода решения, использующего современные вычислительные инструменты, то строгие математические модели программирования могут быть решены за разумное время и, таким образом, использованы для практического принятия решений. В этой статье мы изучаем формулировку MIP Prasad и Maravelias (2007), потому что это формулировка MIP, которая может быть использована для решения широкого спектра проблем в многостадийных пакетных процессах, но это также одна из самых сложных моделей MIP для пакетного планирования. Тем не менее, мы считаем, что основные идеи метода, разработанного в этой статье, могут быть применены к другим задачам оптимизации процесса.

Остальная часть статьи структурирована следующим образом. В разделе 2 мы формально определяем имеющуюся проблему и представляем формулировку MIP для одновременного пакетирования и планирования в многоступенчатых пакетных процессах. В разделе 3 мы кратко обсудим грид-вычисления, а в разделе 4 представлен предлагаемый алгоритм динамической декомпозиции. В разделе 5 изложены наши результаты расчетов на ряде прототипных примеров этих проблем и приведены рекомендации по наиболее эффективному использованию сетевого компьютера. Статья завершается кратким изложением и некоторыми темами для дальнейшего исследования.

1. **Пакетирование и планирование многоступенчатых пакетных процессов**
   1. **Постановка задачи**

Проблема одновременного пакетирования и планирования может быть выражена следующим образом (см. рисунок 1):

Данны

(i) набор заказов (*i*∈*I*) с высвобождением / временем исполнения , / и спросом ,

(ii) набор единиц обработки (*j*∈*J*) с минимальным / максимальным размерами партии / , временем обработки и стоимостью обработки для заказа *i*,

(iii) набор этапов (*k*∈*K*) с параллельными единицами обработки (*j*∈*J* (*k*); *J* = *J*(1) ∪ *J*(2)… ∪ *J*(|*K*|)) на каждом этапе *k*,

(iv) набор запрещенных единиц *FJ*(*i*) для заказа *i* и набор запрещенных производственных путей (*j*, *j*′)∈*FP* для всех заказов,

определить

(i) количество и размер партий, необходимых для выполнения каждого заказа (решение о партии),

(ii) распределение партий по единицам обработки на каждом этапе,

(iii) последовательность назначенных партий в каждой единице обработки,

чтобы минимизировать время, необходимое для выполнения всех заказов.

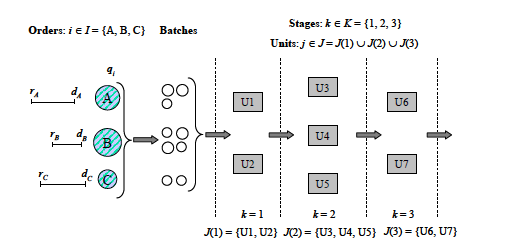


Рисунок 1: Пример многоступенчатой периодической установки.

Предполагается, что все заказы проходят все этапы, что для промежуточных звеньев между этапами доступно неограниченное хранилище, а время переключения незначительно.

**2.2 Формулировка MIP**

Чтобы учесть пакетные решения, нам сначала нужно вычислить минимальное () и максимальное () возможное количество партий, на которые можно разделить порядок *i*∈*I*:



Где это самый большой размер партии для заказа *i*, который может быть обработан на всех разрешенных единицах, и это набор единиц измерения, который можно использовать для обработки заказа *i* на этапе *k* (подробнее см. Prasad and Maravelias (2007)).

Как только мы вычислили , мы постулируем множество потенциальных пакетов *L*(*i*) = {1, 2,… } для каждого порядка *i*∈*I*. Таким образом, партия обозначается парой (*i*, *l*): индекс *i* обозначает порядок, которому соответствует партия, а индекс *l*∈*L*(*i*) обозначает номер партии. Три основных решения для одновременного пакетирования и планирования многоступенчатого процесса моделируются следующим образом:

(i) Выбор партии: = 1, если активна l-я потенциальная партия (т.е. выбрана для удовлетворения заказа *i*)

(ii) Назначение партии: = 1, если партия (*i*, *l*) назначена блоку обработки *j*

(iii) Последовательность партий: = 1, если партия (*i’,l’*) обрабатывается после (*i*, *l*) на той же единице этапа *k*. Набор пакетов (*i*, *l*) и (*i’,l’*), которые могут быть упорядочены на единицу, обозначается *IL*:



Чтобы уменьшить количество симметричных решений, выбор партий осуществляется в числовой последовательности:



Очевидно, что размер пакета (*i*, *l*) может быть ненулевым, только если соответствующий двоичный активен,



Выбранная партия должна обрабатываться ровно один раз на каждом этапе,



Для каждого заказа *i* размеры выбранных партий должны удовлетворять спросу на заказ *i*:



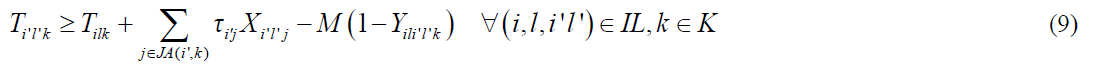
На любом этапе размер пакета (*i*, *l*) должен находиться в пределах минимальной и максимальной производительности единиц обработки, где обрабатывается партия (*i*, *l*),



Последовательность между партиями (*i*, *l*) и (*i*’, *l*’) применяется через (8), когда обе партии назначаются одному и тому же устройству на этапе *k*:

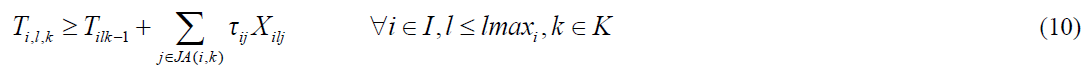


Если партия (*i*', l') обрабатывается после партии (*i*, *l*), то (9) обеспечивает соответствующую задержку,

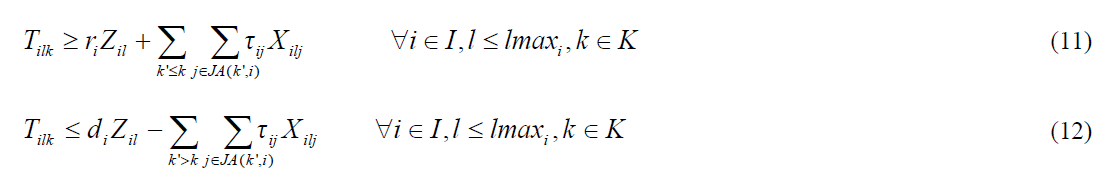


где - время окончания партии (*i*, *l*) на этапе *k*.

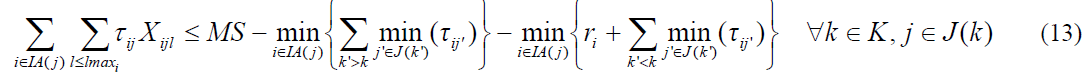
Кроме того, партия может быть обработана на стадии *k* только после того, как она закончена на стадии *k*-1:



Освобождение и ограничения по времени применяются через (11) (для *k* = 1) и (12) (для *k* = |*K*|), соответственно. Кроме того, (11) и (12) обеспечивают действительную нижнюю и верхнюю границу для :

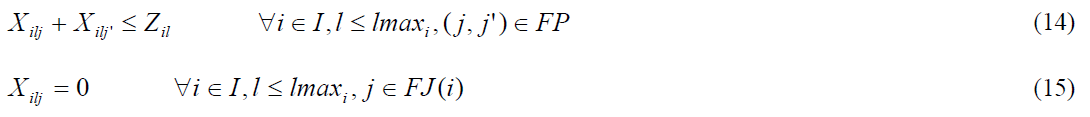


Для того, чтобы сумма времен обработки пакетов, назначенных единице, не превышала временное окно, доступное для обработки, (13) исключает ряд недопустимых назначений.



где *IA*(*j*) = *I*\ *FI*(*J*) - набор заказов, которые могут быть назначены единице *j*.

Запрещенные пути и запрещенные назначения применяются через (14) и (15) соответственно.



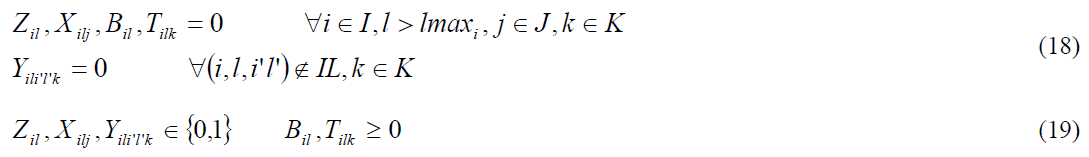
Очевидно, что для удовлетворения спроса на заказ требуются как минимум партии . Поскольку выбор партий выполняется последовательно через (3), двоичные файлы фиксируются на 1 для *l*≤:



Чтобы уменьшить размер ветви и связанного дерева, мы избегаем симметричных решений, добавляя следующее ограничение:



Очевидно, что партии (*i*, *l*) с *l*> i не могут быть выбраны, назначены и упорядочены



В этой статье мы заинтересованы в минимизации makepan, целевой функции, которая приводит к сложнейшим моделям MIP:



где makepan MS должен быть больше, чем время окончания всех партий на последнем этапе,



Модель MIP **P**, которую мы изучаем в этой статье, состоит из (3) - (21). Здесь мы хотели бы подчеркнуть, что для данной партии оборудования (то есть фиксированного набора этапов K и фиксированного набора блоков обработки J) размер модели P зависит от количества заказов и количества партий. Обратите внимание, что все переменные и большинство ограничений определены для *i*∈*I* и *l*≤. Поэтому для набора заказов величина неявно определяет размер формулировки посредством вычисления (см. (2))..

Модель P может использоваться для решения широкого круга проблем (например, максимизация производства за фиксированный временной интервал, минимизация запаздывания / простоя), и было показано, что она дает лучшие решения, чем существующие методы. Однако введение дополнительных переменных решения привело к увеличению размера и сложности формулировки. Чтобы улучшить свое решение, Prasad и Maravelias (2007) представили алгоритм предварительной обработки, который использует информацию временного окна, чтобы исправить некоторые переменные последовательности и класс ужесточающих неравенств. Тем не менее, модель P, как и большинство моделей MIP для планирования процессов, в настоящее время не может использоваться для решения практических задач в течение нескольких часов времени настенных часов.

1. **Cеточные вычисления**

Мы внедрили наш подход к решению этой проблемы в рамках GAMS, используя параметры сетки, которые описаны в Bussieck и др. (2007).

Модель вычислений, которая очень полезна для приложений оптимизации, - это парадигма мастер-работник, и именно эту модель мы принимаем. Главный процессор запускает процесс GAMS, который создает и порождает все задачи, а также собирает результаты каждой задачи. При использовании GAMS/grid каждая задача создается главным каталогом в отдельном каталоге. Главный процесс порождает задачу для выполнения на работнике, и наличие файла «выполнено» в каталоге задач означает, что задача была выполнена. Хотя написание файла решения может занять много времени, создание «готового» файла может быть выполнено практически мгновенно. Новая задача создается (с помощью скрипта) каждый раз, когда GAMS встречает оператор решения в файле модели. Наличие «готового» файла и загрузка решения могут быть выполнены в GAMS с помощью примитива «handlecollect». Эта простая конструкция означает, что механизм GAMS/grid может работать с различными механизмами grid, так как вышеупомянутый сценарий может быть легко адаптирован для конкретного механизма grid. Кроме того, расширения сетки являются частью стандартного языка моделирования GAMS и поэтому доступны для использования в любой модели, написанной на этом языке.

Конкретный механизм сетки, который мы используем, предоставляется менеджером ресурсов Condor (Epema et al., 1996; Litzkow et al., 1988) и включает в себя большой набор машин, работающих под управлением операционной системы Linux. Наша реализация не требует, чтобы работники совместно использовали файловую систему с мастером, поскольку это значительно ограничило бы число работников, которые могли бы использоваться. Вместо этого Condor отправляет каталог задач в «песочницу» на рабочем компьютере, на котором выполняется задача. Только файлы, метка времени которых была обновлена, отправляются обратно на отправляющую машину. Все эти детали скрыты от моделиста; они выполняются автоматически, когда оператор решения встречается после однострочной директивы в GAMS для использования параметра сетки:



Дополнительные примеры синтаксиса GAMS, используемого для представления сетки, и базовые методы для работы с различными механизмами сетки можно найти в Bussieck et al. (2007).

Для эффективности вычислений важно, чтобы мастер мог передавать обновленную информацию работнику, который в данный момент выполняет задачу. Например, мастер может узнать новое действующее значение при параллельной обработке большой ветви и связанного дерева. GAMS/CPLEX позволяет обновлять существующий набор параметров при выполнении оптимизации с использованием механизма запуска файлов; всякий раз, когда исполняющий процесс CPLEX видит файл триггера, он удаляет файл триггера и считывает и обрабатывает новый файл опций (это аналогично механизму прерывания, который позволяет интерактивно обновлять опции для запущенного процесса решения). Однако в нашем случае процесс CPLEX выполняется в «песочнице» на рабочем компьютере, и мастер-процесс может создавать файлы только в исходном «каталоге задач». Наша реализация решает эту проблему с помощью утилиты «condor\_chirp».

Программа «condor\_chirp» имеет три варианта: получить, удалить и поставить. Вместо того, чтобы просто запускать процесс CPLEX на рабочем месте, дополнительный рабочий процесс также выполняется на рабочем, который неоднократно вызывает condor\_chirp. Вспомогательный процесс постоянно выполняется в цикле, периодически проверяя наличие файла триггера на главном сервере (используя выборку), и копирует новый файл параметров, если файл триггера существует. (Обратите внимание, что файл триггера на главном сервере удаляется condor\_chirp до того, как файл опций извлекается, чтобы избежать условий гонки.) Вспомогательный процесс также помещает вновь найденные действующие решения (выгруженные из запущенного процесса CPLEX с использованием средства GAMS/BCH) из рабочей в главную копию каталога задач. Главный процесс может отслеживать эти файлы, чтобы определить, является ли новый сотрудник, работающий на рабочем месте, лучше своего текущего сотрудника. Таким образом, основной процесс GAMS узнает о хороших решениях гораздо раньше, чем если бы он ожидал завершения рабочей задачи.

Обратите внимание, что вспомогательный процесс и condor\_chirp не нужны, если главный и рабочий процессы совместно используют файловую систему. Эти утилиты просто позволяют нам имитировать функциональность, которая необходима нашему методу оптимизации, из такой общей файловой системы. Однако, поскольку мы не требуем, чтобы наши сотрудники совместно использовали файловую систему, число работников, которые мы обычно можем использовать в любой момент времени, значительно превышает 1000. Несмотря на то, что это вычислительный ресурс, совместно используемый сообществом пользователей, мы, как правило, можем обслуживать более 600 сотрудников в течение длительных периодов времени. Можно использовать гетерогенные машины в качестве рабочих, и мы реализовали это для схемы сетки, состоящей из машин Solaris, Windows и Linux. Однако в приведенных здесь результатах все работники использовали одну и ту же операционную систему.

1. **Алгоритм декомпозиции**

Целью метода решения, описанного в этом разделе, является последовательная динамическая декомпозиция задачи P на подзадачи разных уровней сложности. Во-первых, модель P раскладывается на набор подзадач 1-го уровня, где каждая подзадача имеет фиксированное количество пакетов. Во-вторых, если подзадача 1-го уровня остается нерешенной после достижения предопределенного предела ресурса, она разбивается на набор подзадач 2-го уровня с фиксированными назначениями пакетов для всех пакетов на данном этапе. Если подзадача 2-го уровня не решается в рамках заданного ограничения ресурсов, то она дополнительно разбивается на набор подзадач 3-го уровня на основе назначения пакетов на втором этапе. Процесс может повторяться до тех пор, пока проблема P не будет разложена на подзадачи, которые могут быть легко решены или сокращены. Каждая подзадача является задачей, созданной главным процессом и порожденной для выполнения на рабочем месте. Детали алгоритма представлены в следующих двух подразделах, в то время как схема алгоритма представлена ​​на рисунке 2. Псевдокод представлен в Приложении А. Иллюстративный пример алгоритма разложения представлен в Приложении B.

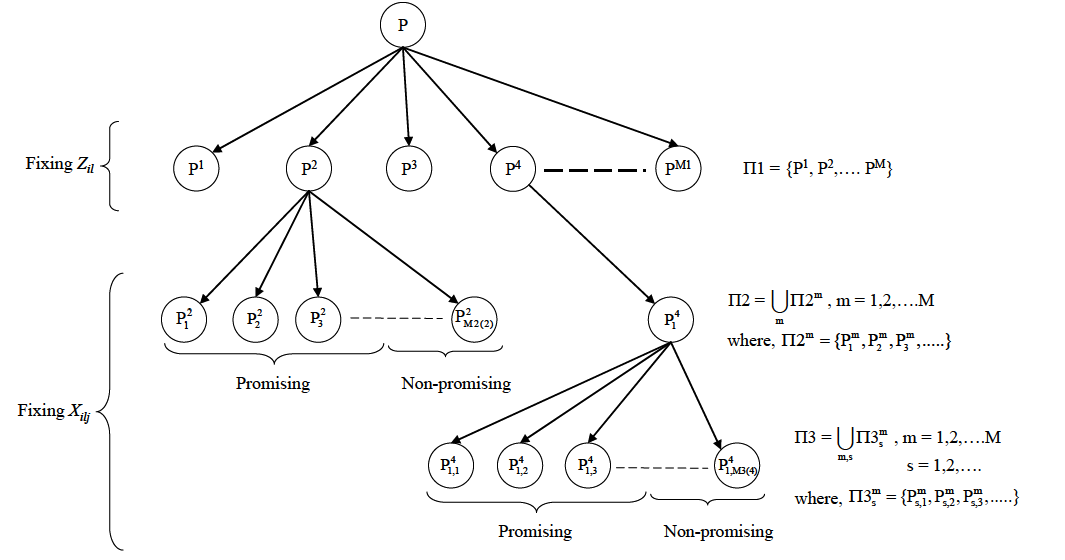


Рисунок 2 - Принципиальная схема алгоритма разложения.

* 1. **Фиксирование количества партий**

На первом уровне модель P заменяется набором Π1 подзадач *M1* {, ,…, , }. Каждая подзадача ∈Π1 имеет фиксированное количество пакетов, т.е. соответствует уникальному пакетному решению. Таким образом, если - количество партий для порядка I ∈ I в решении для пакетирования, то подзадача соответствует комбинации {, ,…} партий с ≤ ≤ , ∈I. Каждая подзадача 1-го уровня генерируется установкой,



в уравнениях (3) - (21) модели Р.

Обратите внимание, что если (22) верно, то все переменные фиксируются через (16) и (18). Следовательно, каждая подзадача 1-го уровня соответствует многоэтапной задаче планирования только с решениями о назначении и последовательности. Число подзадач 1-го уровня равно , а количество пакетов в подзадаче равно . В нашей реализации мы порождаем все проблемы M1 с коротким ограничением по времени, поскольку мы видим, что почти все эти модели по существу просты для решения, и только один или два из них создают трудности для решателя. Кроме того, исправление Z-переменных позволяет выполнить некоторые упрощения в модели.

* 1. **Исправление назначений единиц пакета**

Хотя большинство подзадач 1-го уровня либо неосуществимы, либо сокращены в течение одной минуты, некоторые из них не могут быть решены до оптимального уровня в рамках данного лимита ресурса. Эти подзадачи разбиты на подзадачи 2-го уровня. В частности, каждая сложная подзадача разбивается на множество = {, ,…, } из *M2*(*m*) подзадач 2-го уровня (обратите внимание, что верхний индекс обозначает родительскую подзадачу). Разложение на этом уровне основано на назначении партий единицам на одной стадии . В этой статье мы используем назначения на этапе «узкого места», то есть на этапе с наибольшим отношением среднего времени обработки к числу обработок. Цель здесь двоякая:

a) Определить подмножество назначений пакетных единиц на этапе , которые потенциально могут привести к хорошим решениям. Для каждого такого многообещающего задания мы генерируем подзадачу 2-го уровня.

б) Определить подмножества заданий, которые вряд ли приведут к хорошим решениям (не обещающие задания). Для каждого такого подмножества мы генерируем одну подзадачу.

Назначение единиц партии на этапе считается многообещающим, если оно сбалансировано, т.е. если партии равномерно распределены по всем единицам в J(). Рациональное обоснование этого критерия заключается в том, что при минимальном графике выполнения работ все узлы этапа узкого места должны быть равномерно загружены. Если это не так, то лучшее решение можно получить, переместив некоторые партии из сильно загруженного блока в менее загруженный блок. Если бы допускались дробные присвоения, то идеально сбалансированное присваивание было бы таким, где /|J()| партии назначаются каждому подразделению на этапе . Поскольку дробные задания недостижимы и почти сбалансированные задания могут привести к хорошим решениям, мы рассмотрим более широкий набор многообещающих заданий. В частности, мы рассчитываем минимальное и максимальное количество партий, которые могут быть назначены одному блоку на этапе :



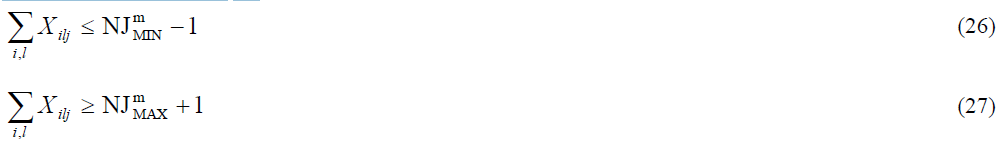
где ∈ [0.8, 1] и ∈ [1.0, 1.2] (в этой статье мы используем = 0.9 и ∈ = 1.1).

Назначения, в которых все подразделения на этапе . имеют от до назначенных партий, считаются многообещающими. Задания, в которых хотя бы у одного из подразделений меньше или больше назначенных партий, считаются не обещающими. Каждое многообещающее задание изучается независимо, то есть создается подзадача 2-го уровня путем добавления следующего ограничения в формулировку родительской подзадачи 1-го уровня:



где D(j) - набор пакетов, которые назначены для блока j в текущей подзадаче 2-го уровня.

Все назначения, которые происходят из-за слегка (или сильно) загруженной единицы, рассматриваются одновременно, то есть для каждого такого поднабора назначений создается одна подзадача 2-го уровня путем добавления (26) или (27) для одной единицы на этапе :



Обратите внимание, что добавление (26) или (27) для одной единицы приводит к одной бесперспективной подзадаче. Таким образом, количество бесперспективных подзадач равно удвоенному количеству единиц на этапе . Если имеется только два блока, одно из двух вышеупомянутых ограничений становится избыточным, что вдвое уменьшает количество подзадач. Ясно, что не обещающие подзадачи имеют большие пространства решений, чем многообещающие, но обычно их легко убрать, поскольку они имеют большие нижние границы. Обратите внимание, что рассматриваются все возможные назначения на этапе .

Если подзадача 2-го уровня не может быть решена в пределах лимита ресурсов, используемых для второго этапа алгоритма, то она может быть дополнительно разложена на набор подзадач 3-го уровня, используя другой этап в качестве этапа узкого места. Таким образом, подзадачи 3-го уровня имеют фиксированное количество пакетов (начиная с 1-го шага) и фиксированные назначения единиц пакета в два этапа. Процесс может повторяться до тех пор, пока все этапы не будут устранены или все подзадачи не будут решены легко.

Наконец, мы разработали процедуру предварительной обработки для определения невыполнимых назначений единиц партии, которые соответствуют многообещающим подзадачам. Хотя такие подзадачи могут быть быстро сокращены, они все же потребляют ресурсы для своего создания и решения. Предложенная процедура расширяет наш алгоритм путем априорного скрининга недопустимых подзадач.

1. **Результаты**

Мы сообщаем об использовании подхода декомпозиции на коллекции из четырех задач. Проблемы становятся все более сложными, они по существу параметризованы через и доступны в формате GAMS или MPS от авторов.

Первая модель может быть решена напрямую с использованием CPLEX версии 10.2 на стандартном рабочем столе. Важно отметить, что мы не навязывали верхние границы для переменных T (например, ), так как в предварительном тестировании мы заметили, что более ограничительную (и другую) проблему легче решить. Для сравнения отметим, что эта проблема решается всего за 17 секунд при исследовании 117 узлов, что дает подтверждение оптимальности со значением 905. Обратите внимание, что для этого решения использовался нестандартный набор опций CPLEX (файл параметров 1).

Вторую модель сложнее решить. Более ранние версии CPLEX не смогли обработать модель за 2 часа на нашем настольном компьютере. Тем не менее, версия 10.2 CPLEX подтверждает оптимальность модели (со значением 935) с использованием 56722 узлов всего за 8 минут, опять же с использованием файла параметров 1. Тем не менее, мы использовали эту проблему для тестирования различных схем для решения сетки.

Мы выдвигаем на первый план некоторые вопросы, которые эта проблема поднимает с нашей техникой решения сетки. Альтернативная схема декомпозиции, которая использует сильные опции ветвления CPLEX в сочетании с опцией dumptree, описана в Bussieck et al. (2007). Эта процедура автоматически разбивает данную проблему на ряд подзадач на основе дерева поиска, созданного CPLEX для заданного набора параметров. Процесс может быть повторен для каждой подзадачи, которая не может быть решена в заданный срок, и может использоваться вместо подхода, специфичного для конкретной области, описанного выше. Ключевой вопрос заключается в том, как автоматическая схема сравнивается со схемой для конкретного домена?

Сначала мы попытаемся настроить автоматическую схему максимально эффективно. Основываясь на размере доступного нам грид-механизма, мы разбили исходную модель, используя сильное ветвление, на 400 подзадач. Все подзадачи были созданы для решения в сетке с ограничением по времени, и через два часа все были выполнены при использовании файла параметров 1. (Обратите внимание, что было еще 5 подзадач, которые не были выполнены через два часа, если мы используем параметры CPLEX по умолчанию). В общей сложности 13 часов процессорного времени было затрачено на сетку, но чуть более 2 часов времени настенных часов, так как мы смогли порождать подзадачи и заставить ресурсы сетки работать почти мгновенно. Неэффективность этой схемы подчеркивается тем фактом, что всего было исследовано 2905742 узла по сравнению с 56722, использованными в последовательном решении. Проблема двоякая. Во-первых, сильные варианты ветвления не позволяют CPLEX находить хорошее решение или предельное значение, поэтому 400 сгенерированных подзадач обычно очень похожи на исходную модель. Во-вторых, многие из подзадач работают на узлах, которые могли бы быть поняты, если бы общему лучшему сотруднику было сообщено ранее.

Создание хорошего значения отсечки в корневом узле перед выполнением сильного ветвления значительно повышает производительность. Используя файл параметров, который выполняет эвристику локального ветвления (Danna et al., 2005; Fischetti and Lodi, 2003), генерируется разумное действующее значение менее чем за 30 секунд, которое можно передать как методу сильного ветвления, так и получающимся в результате подзадачам. При этой настройке 30 секунд использовалось для генерации значения отсечки, 530 секунд для сильного ветвления, что привело к 109 подзадачам. Для решения одной из этих подзадач потребовалось 700 секунд (и в целом мы использовали 51 минуту процессорного времени в сетке для исследования 133078 узлов). Таким образом, время настенных часов составило чуть более 21 минуты.

Как это соотносится с конкретной доменной схемой? Очень легко определить, что единственная установка для переменных использует процедуру ветвления 1-го уровня. Когда эти значения являются фиксированными, следует процедура, описанная выше, сильное ветвление заняло менее 300 секунд, в результате чего было выполнено 48 заданий, самое длинное из которых заняло всего 100 секунд (в общей сложности вычисление сетки заняло менее 5 минут, и был исследован только 9601 узел). Таким образом, проблема может быть решена примерно за 7,5 минут времени настенных часов. Таким образом, знание предметной области здесь эффективно, по крайней мере, в сетке.

Понятно, что сеточное решение вышеуказанных проблем не так эффективно, как решение с использованием серийной версии CPLEX с тщательно подобранными параметрами. Тем не менее, для оставшихся двух проблем, которые мы обсуждаем здесь, последовательный вариант не может решить проблему за два часа (или даже с гораздо более длительным временем). Например, найденный экземпляр проблемы 3 определил нижнюю границу для 1176 через два часа с действующим значением 1186, в то время как для экземпляра проблемы 4 соответствующие значения были 1395 и 1416.

Экземпляр задачи 3 гораздо сложнее, чем проблема 2. Использование наилучшего подхода с сеткой из нашего исследования проблемы 2 позволило получить существующее значение 1185 с эвристикой локального ветвления, которая при передаче в процедуру сильного ветвления породила 390 проблем примерно за 800 секунд. Через два часа 93 подзадачи все еще не были завершены; наихудшая нижняя граница осталась на уровне 1170. Интересно отметить, что всего за 3 часа настенных часов сеточная система доставила более 8 дней процессорного времени и позволила изучить 36775351 узлов в целом. Однако этот подход не решает нашу проблему, мотивируя использование подхода динамической декомпозиции доменов, который был изложен выше.

Мы сообщаем о четырех различных подходах динамической декомпозиции. В каждом из подходов мы показываем эффект от использования все большего количества специфичных для предметной области знаний при разделении (т. е. больше уровней с использованием подхода, описанного выше). Однако обратите внимание, что все процедуры полностью автоматизированы либо с помощью опций CPLEX, либо с помощью реализации в GAMS описанного выше алгоритма.

Первый подход просто использует метод сильного ветвления для создания раздела, за которым следует решение с использованием файла параметров 1 в течение часа с последующим перераспределением любых подзадач, которые не были полностью обработаны при достижении этого временного предела, снова с использованием сильного ветвления. Хотя можно было бы поэкспериментировать с другими вариантами ограничения ресурса, кажется, что 1 час - это хороший компромисс, который позволит нам использовать несколько уровней в процедуре решения, а также даст CPLEX достаточно времени для обработки как можно большего количества сложных подзадач. Этот подход не позволил сгенерировать ряд различных вариантов подзадач в процедуре сильного ветвления - общее число созданных подзадач фактически заполнило диск компьютера-отправителя.

Второй подход использует разделение на домены на 1-м и 2-м уровнях. При таком разделении на 2-м уровне было создано 654 подзадачи, из которых 6 остались через час с наихудшей нижней границей 1176. Эти 6 были автоматически разделены с использованием метода сильного ветвления, что привело к появлению еще 1560 подзадач, что привело к тому, что 11 осталось еще через час с наихудшей нижней границей без изменений. Повторение этого процесса еще два раза привело к 15 невыполненным подзадачам после того, как было израсходовано почти 12 дней процессорного времени на 17659 подзадач и 128926860 узлов. К сожалению, наихудшая нижняя граница осталась на уровне 1176 после этих вычислений. Мы приходим к выводу, что для решения этой проблемы требуется более двух уровней доменных разделов.

Третий подход использует разделение на конкретные области на 1-м, 2-м и 3-м уровнях в сочетании с ограничением ресурсов для решения подзадачи в 1 час. Любые подзадачи, которые остаются нерешенными после этих двух уровней вычислений, обрабатываются с помощью процедуры сильного ветвления. Таким образом, мы используем привязку к домену в верхней части дерева поиска, после чего следует полностью автоматическая схема. Как и выше, после разбиения 2-го уровня осталось 6 подзадач, но метод, специфичный для предметной области, позволил решить 56628 подзадач на 3-м уровне. В общей сложности 26 подзадач осталось после попытки трех уровней. Схема автоматического сильного ветвления была применена к этим остающимся проблемам и использовала еще 3 перераспределения в некоторых случаях, обрабатывая в общей сложности еще 31761 подзадачу. В целом было использовано 17,5 дней ЦП и обработано 222065793 узла. Мало того, что все подзадачи были решены на ресурсах сетки, но также выполнение метода сильного ветвления было выполнено на сетке. Перераспределение домена является простым кодом GAMS и, таким образом, было выполнено на отправляющем компьютере. Учитывая время генерации и время решения сетки, эта проблема была решена всего за 9 часов времени настенных часов. Оптимальное решение - 1185.

Последний подход использует привязку к четырем уровням, зависящую от предметной области, в результате чего все переменные фиксируются. В этом случае, возникающие в результате подзадачи, фактически являются проблемами планирования и, возможно, могут быть решены более эффективно с использованием специального кода. Всего было решено 103884 MIP-подзадач (по сравнению с 88389 выше), но это заняло 36,75 дней процессорного времени и было обработано 614633448 узлов. Однако все подзадачи были решены после четырехуровневой декомпозиции. Время настенных часов для всего процесса составило чуть более 12 часов.

Ясно, что подход динамической декомпозиции области может использоваться для оптимального решения данных задач за 9 - 12 часов при условии наличия компьютера с большой сеткой. Следует отметить, что нам часто удавалось иметь более 600 рабочих, одновременно работающих над созданными нами подзадачами, поскольку мы реализовали наш интерфейс с Condor без требований общей файловой системы. Это также тот случай, когда большая часть усилий связана с улучшением нижней границы и что локальная эвристика ветвления CPLEX очень эффективна для быстрого поиска решений, близких к оптимальным. Существует компромисс между количеством разложений, зависящих от предметной области, и автоматическими разложениями с сильным ветвлением. По мере того, как принимается больше решений, автоматизированный подход сильного ветвления становится более привлекательным. Однако, когда иерархическая структура является частью структуры проблемы, также ясно, что существуют значительные вычислительные преимущества для использования такой структуры.

Проблема 4 остается нерешенной в настоящее время с наиболее близким решением со значением 1416 и нижней границей 1407. При декомпозиции проблемы на 2-й уровень возникают 745 подзадач, из которых 3 остаются через 1 час с наихудшей нижней границей на уровне 1407. Одна из этих нерешенных подзадач была разделена на 3-й уровень на 28886 подзадач. Из них 29 были оставлены через 1 час, и худшая нижняя граница не изменилась. Использование схемы строгого ветвления в этих 29 подзадачах породило еще 9571 подзадачу, многие из которых не удалось решить через 1 час. Фактически, примерно за 12 часов времени настенных часов на эти подзадачи было затрачено около 126 дней процессорного времени, и было исследовано более 854142966 узлов. Понятно, что для решения этой проблемы в требуемые сроки (настенные часы по 10-20 часов) потребуются более сложные подходы.

1. **Выводы**

В этой статье мы рассмотрели решение проблемы одновременного пакетирования и планирования для многоступенчатых пакетных процессов. Эта проблема важна в широком спектре областей применения, и во многих случаях необходимо ее решение в течение ограниченного периода времени. Формулировка, которую мы рассматриваем, является полностью общей и предоставляет решения лучшего качества, чем конкретные специализации, но, как правило, очень требовательна с точки зрения вычислительных усилий, и многие практически размерные примеры выходят за рамки современных методов решения.

Наш подход включает в себя вычисление сетки, и мы используем большой механизм сетки, предоставляемый и управляемый системой Condor. Тщательно ограничивая наши потребности в связи и используя средства для снижения требований к нашим машинам, мы можем получить доступ к большому количеству вычислительных машин для этого приложения.

Статическое разложение в сочетании с этими большими объемами вычислительных ресурсов, по-видимому, неэффективно для этих проблем. Автоматическая динамическая процедура, основанная исключительно на сильном ветвлении, кажется более мощной, особенно когда предоставляется хорошее значение отсечения, но она не может решить более сложные случаи.

Динамическая декомпозиция, основанная на знании предметной области, является наиболее эффективной схемой для разделения проблемы, и в сочетании с большим количеством ресурсов сетки позволяет решать более крупные, более практичные модели этого типа. В документе содержатся некоторые рекомендации о том, когда переходить от декомпозиции знаний в области к автоматической, основанной на сильном ветвлении, но в этой области требуется дальнейшая работа. Мы считаем, что наши подходы обобщаемы для многих других проблемных классов.

Несмотря на то, что этот подход может решить проблемы большего размера во время, практичное для их применения, ясно, что этот класс моделей может обеспечить тестовые задачи, которые будут продолжать бросать вызов нашим оптимизациям, моделированию и вычислительным ресурсам в обозримом будущем.

**Приложение А**

Алгоритмы генерации подзадач 1-го и 2-го уровня приведены в таблицах А1 и А2 соответственно. Первоначально подзадачи 1-го уровня создаются на рабочих компьютерах и решаются с ограничением ресурсов. Набор не решенных подзадач 1-го уровня идентифицируется и разбивается на подзадачи 2-го уровня с использованием алгоритма из таблицы A2. Затем эти подзадачи 2-го уровня создаются на рабочих компьютерах. Если есть подзадачи 2-го уровня, не решенные в пределах лимита ресурсов, их можно дополнительно разложить, исправив двоичные файлы на другом этапе.

Таблица А1- Алгоритм генерации подзадач 1-го уровня

0 ПОЛУЧИТЬ и

Инициализировать: m = 1

1 Выбрать новую комбинацию m номеров партий

ДЛЯ всех заказов

Выбрать количество партий таких, что

Для комбинации m рассчитать:

2 Создать подзадачу 1-го уровня для комбинации m

Установить в

3 ЕСЛИ больше нет новой комбинации номеров партий

КОНЕЦ

ИНАЧЕ

Увеличить m на 1

Перейти на 1

Таблица А2 - Алгоритм генерации подзадач 2-го уровня

0 Выберите этап для исправления

ПОЛУЧИТЬ J() и

Рассчитать и используя (23) и (24)

1 ДЛЯ всех единиц в

Выберите количество партий, которые будут назначены между и

ЕСЛИ сумма номеров партий ≠

Перейти к 4

ИНАЧЕ

2 ДЛЯ всех единиц в

Выберите партии, которые будут назначены на текущий блок

ЕСЛИ текущая комбинация партии-единицы недопустима (см. §4.3)

Перейти к 2

ИНАЧЕ

3 Генерация подзадачи 2-го уровня

Добавьте (25) к модели родительской подзадачи

ЕСЛИ больше нет комбинации партий в единицах

Перейти к 1

ИНАЧЕ

Перейти к 2

4 ЕСЛИ комбинация номеров партий больше не будет назначена единице

Перейти к 5

ИНАЧЕ

Перейти к 1

5 ДЛЯ всех единиц в

Сгенерировать не обещающую подзадачу второго уровня

Добавьте (26) или (27) к модели родительской подзадачи

КОНЕЦ

**Приложение Б**

Рассмотрим пример с двумя заказами, двумя этапами и двумя единицами за этап. В таблице Б1 приведены минимальные и максимальные размеры партий в обеих стадиях. В таблице Б2 приведены требования и минимальные и максимальные номера партий для двух заказов. Различные комбинации номеров партий для заказов: (1, 2), (1, 3), (2, 2) и (2, 3). Для каждой из этих комбинаций мы генерируем подзадачу 1-го уровня (см. рисунок Б1). Например, подзадача для комбинации (1, 2) генерируется путем установки = = 1 и = = 2. Таким образом, проблема P в полном пространстве приводит к 4 различным проблемам многоступенчатого планирования, где число партий для каждого заказа фиксировано.

Рассмотрим подзадачу 1-го уровня с комбинацией количества партий как (2, 2), то есть = 4. Предположим, что этап 1 является этапом узкого места . Эти 4 партии должны быть назначены U1 и U2 единицам . Используя = 0,9 и = 1,1, мы вычисляем = 1 и = 3 из (23) и (24). Затем юниты могут иметь три различные возможные комбинации сбалансированных назначений: 1 партия в U1 и 3 партии в U2, или по 2 партии в U1 и U2, или 3 партии в U1 и 1 партия в U2. Первая комбинация приводит к 4 (.) многообещающим подзадачам 2-го уровня. Аналогично, две другие комбинации приведут к 6 (.) и 4 (.) многообещающим подзадачам 2-го уровня соответственно.

Таблица Б1 - Минимальный и максимальный размер партии единиц (юнитов) (в кг)

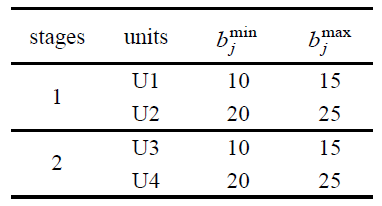
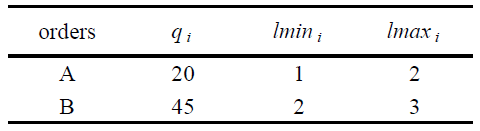


Таблица Б2 - Требования (в кг) и минимальные и максимальные номера партий для заказов



Чтобы получить не перспективные подзадачи и, мы добавим (26) к для единиц U1 и U2 соответственно. Поскольку число единиц в равно двум, должно быть достаточно либо (26), либо (27). Следовательно, подзадача 1-го уровня заменяется шестнадцатью подзадачами 2-го уровня (, , …,), из которых первые 14 являются многообещающими, а последние 2 - не многообещающими. Каждая из подзадач 2-го уровня может быть дополнительно разложена путем фиксации переменных на этапе 2 (см. рисунок Б1).

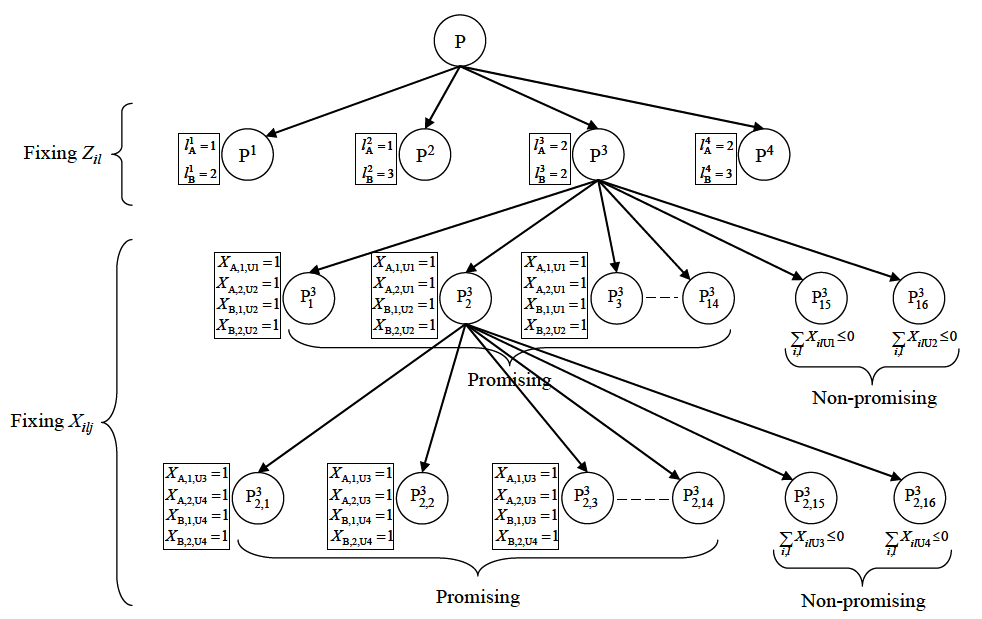


Рисунок Б1 - Генерация подзадач уровня 1 и уровня 2 для иллюстративного примера

**Подтверждения**

Это исследование частично поддерживается Американским химическим обществом - Фонд исследований нефти при гранте PRF № 44050-G9, Управление научных исследований ВВС по гранту FA9550-07-1-0389 и Гранты Национального научного фонда CTS-0547443, DMI-0521953, DMS-0427689 и IIS-0511905.