КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2015 Т. 7 № 3 С. 719–725

Ки&М

СЕКЦИОННЫЕ ДОКЛАДЫ

УДК: 004.023

Предварительная декомпозиция задач дискретной оптимизации для ускорения алгоритма ветвей и границ в распределенной вычислительной среде

С. А. Смирнов[a], В. В. Волошинов

Институт проблем передачи информации им. А. А. Харкевича РАН,
Россия, 127051, г. Москва, Большой Каретный переулок, д. 19, стр. 1

E-mail: [a] sasmir@gmail.com

Получено 30 сентября 2014 г.

В работе рассматриваются возможности реализации крупноблочных схем метода ветвей и границ для решения частично целочисленных задач линейного программирования. В качестве основы берется пакет оптимизации с открытым исходным кодом CBC. Анализируется возможность использования пакета для реализации крупноблочной схемы метода ветвей и границ. Система реализуется с использованием языка Erlang. Проводятся численные эксперименты на основе задачи о коммивояжере, показывающие заметное ускорение распределенной схемы решения задачи по сравнению с единичным однопоточным экземпляром пакета.

Ключевые слова: метод ветвей и границ, крупнозернистый параллелизм

Pre-decomposition of discrete optimization problems to speed up the branch and bound method in a distributed computing environment

S. A. Smirnov, V. V. Voloshinov

Institute for Information Transmission Problems of the Russian Academy of Science, Kharkevich Institute, 19/1 Bolshoy Karetny per., Moscow, 127051, Russia

The paper presents an implementation of branch and bound algorithm employing coarse grained parallelism. The system is based on CBC (COIN-OR branch and cut) open-source MIP solver and inter-process communication capabilities of Erlang. Numerical results show noticeable speedup in comparison to single-threaded CBC instance.

Keywords: branch and bound algorithm, coarse grained parallelism

Работа выполнена при финансовой поддержке программы Президиума РАН № 14 «Проблемы создания национальной научной распределенной информационно-вычислительной среды на основе grid-технологий, облачных вычислений и современных телекоммуникационных сетей».

Citation: *Computer Research and Modeling*, 2015, vol. 7, no. 3, pp. 719–725 (Russian).

Введение

Для решения трудоемких задач дискретной оптимизации широкое распространение получили комбинаторные методы, среди которых наиболее часто встречаются методы ветвей и границ (МВГ). МВГ можно представить в виде процесса последовательного разбиения множества допустимых решений на подмножества с последующим отсечением не содержащих оптимальное решение подмножеств. Его реализация требует обхода некоторого дерева поиска. Каждому узлу дерева соответствует вспомогательная (оценочная) задача, полученная ослаблением части ограничений исходной задачи. Работа алгоритма может существенно зависеть от порядка выбора узлов дерева (правил его обхода). Традиционно ускорение работы МВГ достигается за счет применения механизма параллельных вычислений, суть которого заключается в распределенной обработке оценочных задач некоторым пулом солверов (так называемом мелкозернистое распараллеливание). На нижнем уровне распараллеливанию подвергаются отдельные части алгоритма, однако взаимодействие их не меняется и параллельный алгоритм в целом ведет себя так же, как последовательный. Однако практическое применение такого подхода даже на базе специализированных программных инструментариев требует привлечения квалифицированных программистов, владеющих навыками работы с технологиями параллельных вычислений MPI и/или OpenMP.

Это обстоятельство требует исследовать возможности применения иной схемы декомпозиционного решения задачи дискретной оптимизации. Речь идет о гораздо реже применяемой, так называемой «крупноблочной» («крупнозернистой») схеме распараллеливания [Попов, 2007]. В ее основе — параллельное решение подзадач с обменом информацией о найденных (в ходе решения подзадач) значениях целевой функции на допустимых решениях (так называемой рекордов). При таком подходе полученные подзадачи могут решаться различными вариантами метода ветвей и границ (поиск «в ширину» или «в глубину» дерева ветвлений, различные эвристики выбора следующей вершины и т. п.). Распараллеливание на этом уровне меняет весь алгоритм в целом, и работа, производимая параллельной версией, порой существенно отличается от работы, производимой последовательной версией: возможно, некоторые ее части вообще не считаются одной из версий, но считаются другой, и наоборот. Несколько активных подзадач могут обрабатываться одновременно, каждая в своем, отдельном процессе. Если один из процессов находит допустимое решение, то соответствующее значение целевой функции (так называемой рекорд) может быть разослано остальным процессам, что позволит им, в принципе, существенно ускорить работу за счет отбрасывания заведомо «неоптимальных» частей исходной задачи. Подобный подход обсуждается в литературе [Попов, 2007; Valente and Mitra, 2008; Bussieck, Ferris, and Meeraus, 2009], но широкого применения пока не получил.

Роль исследователя при этом фактически сводится к поиску подходящего способа первоначальной декомпозиции исходной задачи (предпочтительно в форме программы на языке оптимизационного моделирования) и настройке параметров алгоритма МВГ для набора пакетов дискретной оптимизации, подключенных к распределенной системе. Программная реализация обмена значениями рекордов может оказаться значительно проще, чем для «мелкозернистой» схемы МВГ.

В настоящее время существует ряд пакетов оптимизации, применяющих МВГ для решения частично-целочисленных задач оптимизации: LPsolve, GLPK, CBC, SCIP, Cplex. Первые четыре из них имеют открытый исходный код. Таким образом, естественно желание реализовать на основе таких пакетов крупноблочную схему МВГ. В данной работе сделана попытка реализации крупноблочной схемы МВГ для частично целочисленных задач оптимизации на основе пакета CBC. Представленный в работе алгоритм обеспечивает распределенное решение подзадач на множестве хостов.

Пакет оптимизации CBC

Пакет CBC (COIN-OR branch and cut) [Forres, and Lougee-Heimer, 2005] имеет открытый исходный код на языке C++, разрабатывается в рамках проекта COIN-OR (Computational Infra-

structure for Operations Research) и предназначен для численного решения частично целочисленных задач линейного программирования методами отсечений и ветвей и границ. Использование CBC возможно как через автономное приложение, принимающее данные в форматах AMPL [Fourer, Gay, and Kernighan, 2002], MPS и др., так и в виде встраиваемой библиотеки. CBC может использовать многопоточность для ускорения вычислений и является одним из самых эффективных среди пакетов с открытым исходным кодом [Koch et al., 2011].

В своей документации пакет позиционируется в первую очередь как встраиваемый, хотя и обладающий рудиментарно реализованным автономным приложением. В документации CBC присутствует описание нескольких примеров его встраивания. Необходимый минимум действий состоит в создании экземпляра класса, ответственного за решение задач линейного программирования (обычно для этого применяют пакет CLP), и объекта класса CbcModel, с которым и работает пользователь.

При этом оказывается, что автономное приложение находит решения задач заметно эффективнее, чем встраиваемая версия пакета из примеров. Это объясняется использованием в автономном приложении множества дополнительных приемов и эвристик. Существуют и способы встраивания большей части автономного приложения. Для этого можно использовать функцию callCbc, объявленую в файле CbcModel.hpp исходных кодов CBC. Данная функция позволяет запустить процесс решения задачи, имея строку параметров в том же формате, что и у автономного CBC, а также объект модели CBC, нужным образом настроенный пользователем.

В CBC предусмотрены средства, позволяющие получить или установить текущее значение рекорда. У объектов CbcModel присутствует метод setBestSolution, позволяющий передать пакету оптимизации лучшее на данный момент допустимое решение или только значение целевой функции на этом решении. При этом CBC не будет проверять, является ли это решение на самом деле допустимым. Это свойство важно для обмена значениями рекордов, поступающих от решаемых одновременно подзадач, так как подзадачи часто имеют непересекающиеся множества допустимых решений. Контроль за процессом решения задачи возможен, если установить в CbcModel объект функций обратного вызова, реализующий интерфейс CbcCompare. С его помощью можно узнать об обнаружении нового рекорда, а также подбросить новое значение рекорда, не нарушив при этом работу CBC.

Схема работы алгоритма

Работу крупноблочной схемы распараллеливания, используемой в данной работе, можно разделить на два этапа.

Исполнимый модуль генерации подзадач создает набор AMPL-стабов подзадач на основе пользовательских параметров и AMPL-стаба исходной задачи.

Система на языке Erlang [Cesarini and Thompson, 2009] распределяет подзадачи по вычислительным узлам и обеспечивает обмен значениями рекордов между процессами адаптеров CBC, выполняемыми одновременно, а также аккумулирует файлы журналов и решений. После того как были обработаны все подзадачи, система сохраняет наилучшее из полученных от адаптеров допустимых решений, если оно есть.

Рассмотрим более подробно отдельные составляющие реализованной системы.

Модуль генерации подзадач

В частично целочисленной задаче линейного программирования часть переменных может принимать только целочисленные значения. Создание подзадачи происходит путем выделения подмножества множества допустимых решений исходной задачи. Для этого к исходной задаче можно добавить дополнительное ограничение на целочисленные переменные. Такое ограничение может делить диапазон значений одной переменной на две части и брать только одну из них, полностью фиксировать одну переменную, быть произвольным линейным ограничением

на целочисленные переменные. Добавляя несколько таких ограничений, можно разбить исходную задачу на множество взаимодополняющих подзадач.

Исполнимый модуль генерации подзадач nlmod принимает на вход AMPL-стаб исходной задачи, часть переменных в котором могут принимать только целочисленные значения, и список параметров разбиения, например, вызов

```
nlmod stub.nl split 20 split 21 split 22
```

создаст 8 подзадач из исходной задачи stub.nl, зафиксировав все значения каждой из трех переменных, если целочисленные переменные номер 20, 21 и 22 принимают по два значения (см. рис. 1).

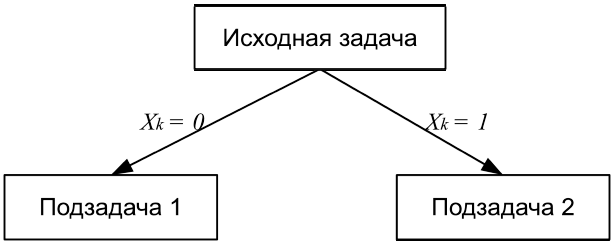

Рис. 1. Разбиение на подзадачи

Адаптер пакета оптимизации CBC

Адаптер пакета оптимизации CBC — это исполнимый модуль на языке C++, обеспечивающий обмен значениями рекордов между пакетом оптимизации и управляющей системой на языке Erlang. Адаптер и управляющая система представлены различными процессами в операционной системе, и поэтому взаимодействие между ними осуществляется с помощью передачи последовательностей байтов через каналы (pipes).

Управляющая система

Управление работой солверов, запускаемых на удаленных хостах, и обмен данными между ними осуществляет система на языке Эрланг. Программы на Эрланге состоят из множества «легких» процессов, взаимодействующих между собой при помощи передачи сообщений. Процессы в системе обмена данными могут быть нескольких типов, названных по имени модуля, содержащего основную часть кода, выполняемого ими.

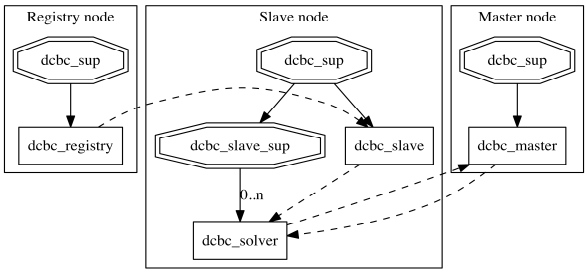

Рис. 2. Зависимости между процессами

Решением задачи управляет процесс dcbc_master, которому при запуске передаются список файлов подзадач и адрес узла, хранящего идентификаторы процессов, осуществляющих запуск пакета CBC на удаленных хостах. Управляющий процесс получает список доступных вычислительных узлов (dcbc_slave) и передает каждому из них один или несколько стабов. Далее вычислительные узлы запускают процессы, управляющие работой CBC (dcbc_solver) через адаптеры CBC. Ниже отдельные компоненты системы рассмотрены более подробно. На рис. 2 представлено дерево супервизоров распределенного приложения. Восьмиугольники — это

процессы-супервизоры, запускающие и контролирующие работу рабочих процессов (прямоугольники), например перезапуская дочерние процессы, завершившиеся аварийно. Стрелки с пунктирными линиями показывают, какие процессы отслеживают состояние других, чтобы выполнить те или иные действия, если отслеживаемый процесс завершился.

Процесс dcbc_solver

«Порты» — это один из способов взаимодействия Эрланга с внешним миром. С их помощью можно запустить внешнюю программу и затем взаимодействовать с ней посредством обмена последовательностями байтов. В свою очередь, внешняя программа может получать и отправлять последовательности байтов, пользуясь парой заданных заранее файловых дескрипторов.

Процессы dcbc_solver — это так называемые процессы-серверы, совершающие определенные действия для каждого полученного сообщения. Основное назначение процессов dcbc_solver состоит в запуске порта к адаптеру СВС с последующим преобразованием форматов между бинарными сообщениями, передаваемыми через порт, и сообщениями, которыми обмениваются процессы в Эрланге. Также данный процесс следит за состоянием адаптера СВС, обнаруживая завершение процесса.

Процесс dcbc_slave

Процессы dcbc_slave запускаются по одному на вычислительный узел. Они обеспечивают запуск процессов dcbc_solver по запросу от управляющего процесса. При этом контролируется число запускаемых процессов, чтобы не допустить перегрузки вычислительного узла.

Процесс dcbc_master

Управляющий процесс реализован модулем dcbc_master. При запуске он получает список имен файлов-стабов и сразу же отдает на выполнение максимально возможное число подзадач, а далее работает как сервер, принимая сообщения от процессов dcbc_solver, работающих на вычислительных узлах. Получив сообщение со значением рекорда, управляющий процесс сравнивает его с текущим и, если рекорд улучшился, рассылает новое значение всем вычислителям. Получив сообщение о завершении одного из вычислителей, управляющий процесс сохраняет решение и отправляет на тот же узел очередную подзадачу. После того как последняя подзадача была обработана, печатается значение текущего рекорда и сохраняется файл с соответствующим рекорду решением.

Обмен значениями рекордов

Улучшив значение рекорда, СВС сообщает об этом через обратный вызов адаптеру, который, в свою очередь, передает сообщение с новым значением рекорда через файловый дескриптор и порт Эрланга своему процессу dcbc_solver, а он получает сообщение с новым значением рекорда и оповещает об этом управляющий процесс (см. рис. 3). Если значение рекорда всей задачи было улучшено, описанная выше схема повторяется в обратном порядке, после чего всем адаптерам становится известно новое значение рекорда.

4. Вычислительный эксперимент

Вычислительные эксперименты проводились на двух вычислительных узлах: 8 потоков на 2 x Intel Xeon E5620 @ 2.40 ГГц, 16 Гб ОЗУ и 4 потока на Intel Core i7-2600K @ 3.40 ГГц, 8 Гб ОЗУ. Всего системе было доступно 12 потоков. Во всех перечисленных ниже запусках пакет

CBC работал в последовательном режиме (без использования встроенной многопоточности). Исходные коды разработанной системы: https://github.com/ssmir/dcbc

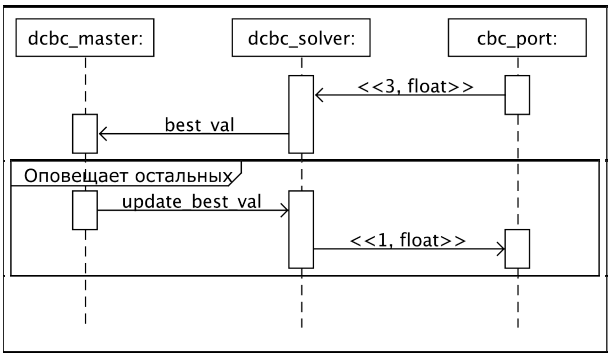

Рис. 3. Обмен рекордами

Тесты проводились на задаче о коммивояжере с числом городов N, равным 80, 90, 100 и 110. Расстояния между городами сгенерированы псевдо-случайным образом.

В качестве ориентира было произведено по одному запуску каждой из исходных задач с однопоточным CBC без какого-либо распараллеливания (время T в таблице 1), а также по одному запуску с пакетом оптимизации SCIP тоже без распараллеливания (время T_s в Таблице 1). Из таблицы видно, что на указанных задачах SCIP работает значительно быстрее CBC. Кроме того, во время тестов было замечено, что многопоточный вариант CBC ведет себя крайне нестабильно, аварийно завершаясь на задачах с N от 100.

Далее было проведено по одному «распределенному» запуску на указанных выше вычислительных ресурсах, результаты перечислены в таблице 2. Здесь M — число зафиксированных переменных, в скобках указано число подзадач, порожденных фиксацией переменных. Фиксировались переменные, соответствующие ребрам наименьшей длины в графе расстояний между городами. Время T_n получено при использовании модуля генерации подзадач, а время T_a — при делении на подзадачи средствами AMPL. Видно, что деление на подзадачи встроенными средствами языка AMPL менее эффективно, чем использование модуля генерации подзадач.

Таблица 1. Запуски исходной задачи

N	T, мин	T_s, мин
80	5,3	1,6
90	20,3	6,0
100	623,8	1,0
110	>10000	74,5

Таблица 2. Разбиение nlmod и AMPL

N	M	T_n, мин	T_a, мин
80	4 (16)	1,5	1,9
90	5 (32)	8,6	11,1
100	6 (64)	1627,0	228,9
110	7 (128)	?	1211,6

Также делались однократные «распределенные» запуски, где варьировалось число фиксируемых переменных (таблицы 3 и 4).

Таблица 3. Запуски для $N = 90$

M	T, мин
0 (1)	20,3
2 (4)	16,0
3 (8)	4,4
4 (16)	8,5
5 (32)	8,6

Таблица 4. Запуски для $N = 100$

M	T, мин
0 (1)	623,8
3 (8)	201,9
6 (64)	1627,0

Выводы

Для задачи коммивояжера представленная в работе схема крупноблочного метода ветвей и границ позволяет получить заметное ускорение по сравнению с однопоточным CBC.

Кроме того, представленная система позволяет решать задачи, на которых не работает многопоточный CBC, завершаясь аварийно.

Однако замечено, что даже в распределенном режиме задача коммивояжера решается не так эффективно, как однопоточным вариантом пакета SCIP. Это требует рассмотреть возможность подключения и SCIP в рассматриваемую в работе систему. Другим возможным направлением развития системы реализация других схем работы, хорошо ложащихся на текущую реализацию, являются, например, запуск пакетов оптимизации не только с различными подзадачами, но и с различными комбинациями настроек солверов, обрабатывающих подзадачи.

Список литературы

Попов Л. Д. Опыт многоуровневого распараллеливания метода ветвей и границ в задачах дискретной оптимизации // Автоматика и телемеханика. — 2007. — № 5. — С. 171–181.

Bussieck M. R., Ferris M.C., and Meeraus A. Grid Enabled Optimization with GAMS}, INFORMS // Journal on Computing. — 2009. — Vol. 21, No. 3. — P. 349–362.

Cesarini F. and Thompson S. Erlang Programming, O'Reilly Media, Inc., 2009.

Forrest J. and Lougee-Heimer R. CBC user guide, INFORMS Tutorials in Operations Research, 2005. — P. 257–277.

Fourer R., Gay D. M., and Kernighan B. W. AMPL: A Modeling Language for Mathematical Programming, second edition, Duxbury Press / Brooks/Cole Publishing Company, 2002.

Koch T. et al. MIPLIB 2010 // Mathematical Programming Computation 3.2. — 2011. — P. 103–163. URL: http://plato.asu.edu/ftp/milpc.html.

Valente P., and Mitra G. A grid-aware MIP solver: Implementation and case studies // Future Generation Computer Systems. — 2008. — Vol. 24, Iss. 2. — P. 133–141.