ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №1(32), 2017, с. 105–119
УДК 004.75

В. В. Волошинов, С. А. Смирнов

Оценка производительности крупноблочного алгоритма метода ветвей и границ в вычислительной среде Everest

Аннотация. В работе исследовался т.н. крупноблочный подход к реализации параллельной работы метода ветвей и границ (МВГ). Исходная задача частично-целочисленного программирования разбивается на несколько подзадач посредством фиксации значений у части целочисленных переменных. Подзадачи решаются параллельно пулом МВГ-решателей. Если в ходе решения подзадач появляется допустимое решение, с наилучшим на данный момент значением целевой функции, то это число рассылается другим решателям. Такой обмен рекордными значениями критерия позволяет взаимно ускорить решение подзадач за счет сокращения перебора вершин дерева ветвлений алгоритма МВГ. Запуск подзадач и обмен данными обеспечивается средствами платформы Everest. В результате тестирования разработанной распределенный системы на случайным образом сгенерированных задачах линейного программирования с частично-булевыми переменными было обнаружено заметное ускорение.

Ключевые слова и фразы: дискретная оптимизация, метод ветвей и границ, крупнозернистый параллелизм.

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 оценочная задача, полученная ослаблением части ограничений (обычно — условий целочисленности переменных) исходной задачи.

Поддержано грантом РФФИ 16-07-01150.

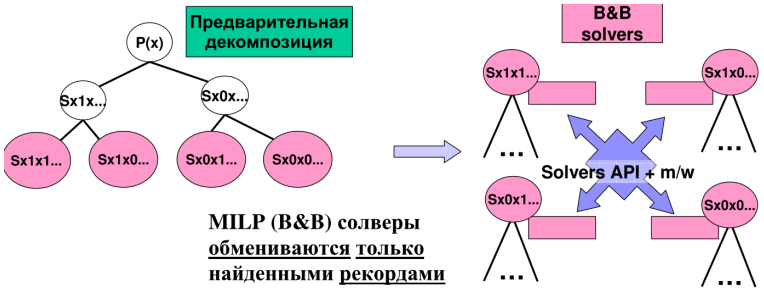

Рис. 1. Принцип реализации «крупноблочной» схемы МВГ в РВС

В работе речь идет об относительно редко применяемой, так называемой «крупноблочной» или «крупнозернистой» схеме распараллеливания [1] (рис. 1). В ее основе — параллельное решение подзадач с обменом информацией о найденных (в ходе решения подзадач) значениях целевой функции на допустимых решениях (так называемых рекордах). При этом несколько подзадач могут обрабатываться одновременно, каждая в своем, отдельном, процессе. Если один из процессов находит допустимое решение, то соответствующее рекордное значение целевой функции может быть разослано остальным процессам, позволяя им, в принципе, существенно ускорить работу, за счет отбрасывания ветвей дерева обхода, заведомо не содержащих оптимального решения. Подобный подход обсуждается в литературе [1–3], но широкого применения пока не получил.

Роль исследователя здесь сводится к поиску подходящего способа первоначальной декомпозиции исходной задачи (например, в форме программы на языке оптимизационного моделирования) и настройке параметров алгоритма МВГ для пакетов дискретной оптимизации, подключенных к распределенной системе. Программная реализация обмена значениями рекордов может оказаться значительно проще, чем для «мелкозернистой» схемы МВГ (рис. 2). В настоящее время существует ряд пакетов оптимизации, применяющих МВГ для решения частично-целочисленных задач оптимизации: LPsolve, GLPK, CBC, SCIP, Cplex, KNITRO, Gurobi и т.д. Первые четыре из них имеют открытый исходный код, что позволяет реализовать на их основе крупноблочную схему МВГ.

В данной работе представлена реализация крупноблочной схемы распараллеливания метода ветвей и границ для частично-целочисленных задач оптимизации. В прошлых работах [4, 5] была описана

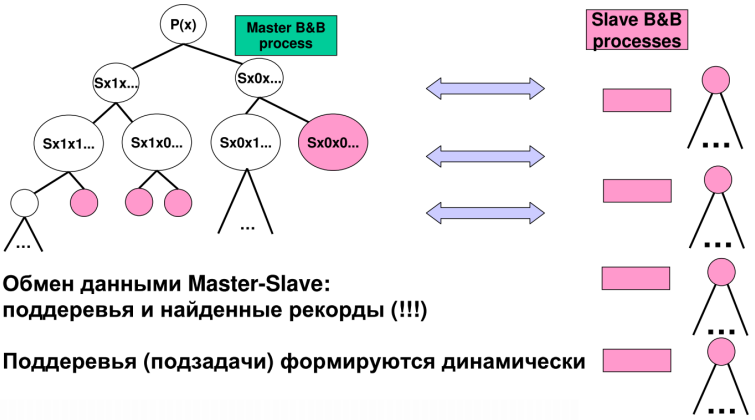

Рис. 2. Принцип реализации «мелкозернистой» схемы МВГ в РВС

система, где коммуникация между процессами-решателями была реализована на языке Erlang. В данной работе Erlang был заменен платформой Everest [6] — достаточно развитым средством развертывания распределенных систем. Данная реализация обеспечивает запуск подзадач на некотором предварительно заданном множестве хостов средствами платформы Everest и обмен рекордами между процессами, решающими подзадачи. Как и ранее, используются пакеты оптимизации с открытым исходным кодом, а именно CBC и SCIP. Представленная система требует предварительного разбиения исходной задачи на подзадачи. Пользователь может сделать это самостоятельно, реализуя некоторый алгоритм разбиения, например, средствами языка оптимизационного моделирования AMPL.

Указанная система была апробирована на классической задаче коммивояжера, сформулированной в виде задачи линейного программирования с частично-булевыми переменными (MILP).

1. Пакет оптимизации SCIP

Пакет SCIP (Solving Constraint Integer Programs) [7] — это один из самых быстрых некоммерческих решателей для задач частично-целочисленного линейного и нелинейного программирования [8]. Решатель представляет собой гибкую и расширяемую систему на языке C. Использование SCIP возможно через консольное приложение scip, поддерживающее множество форматов описания задач. Также

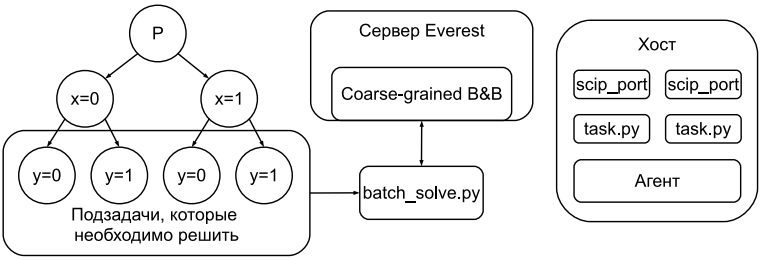

Рис. 3. Архитектура системы

присутствует отдельное приложение `scipampl`, предназначенное для работы с задачами в формате языка моделирования AMPL [9].

Для интеграции в распределенную среду было разработано приложение-адаптер. Адаптер решает две основные задачи: передавать решателю значения рекордов, пришедшие извне, и отслеживать нахождение решателем новых рекордов. Установить новое значение рекорда в решателе можно вызовом `SCIPsetObjlimit`. Вызов приходится делать внутри обратного вызова обработчика событий, так как SCIP не предусматривает безопасных вызовов из нескольких различных потоков. Таким образом, важно установить обработчик на все события, возникающие при работе решателя. Это позволяет минимизировать задержку между получением нового значения рекорда потоком, получающим данные из распределенной системы, и передачей значения данному экземпляру решателя. О нахождении нового рекорда решатель сообщает событием `SCIP_EVENTTYPE_BESTSOLFOUND`. Далее необходимо получить само решение вызовом `SCIPeventGetSol`, а затем получить значение целевой функции вызовом `SCIPgetSolOrigObj`. Важно использовать именно эти вызовы, так как SCIP может трансформировать исходную задачу и тогда значение целевой функции, с которой он работает в настоящий момент, может оказаться непригодным для передачи другим решателям.

2. Реализация крупноблочной схемы

Подготовив набор подзадач, пользователь запускает процесс решения с помощью приложения `batch_solve.py` (рис. 3). Данное приложение формирует архив с начальными данными для сервиса (приложения) Coarse-grained B&B на сервере Everest. Этот сервис имеет заданный по умолчанию набор вычислительных ресурсов, на которых

установлены пакеты CBC и SCIP, а также адаптеры к ним. В результате вызова данного сервиса создается задание (Job), состоящее из множества задач (Task). Каждой подзадаче, сформированной пользователем, соответствует по одной задаче внутри задания. Сервер Everest обеспечивает распределение задач по вычислительным ресурсам (хост на рис. 3). При старте задачи агент запускает специальный скрипт на языке Python, передаваемый в составе входных данных. Данный скрипт `task.py` обеспечивает запуск адаптера решателя `scip_port`, а также взаимодействие с адаптером (через стандартный в Linux механизм обмена данными между процессами *pipes*) и с агентом посредством TCP-сокета, что необходимо для обмена рекордами с другими решателями в рамках одного задания.

Как только решатель находит новый рекорд (допустимое решение, где значение целевой функции меньше уже найденных), об этом узнает адаптер через обратный вызов. Далее, адаптер через пайп передает новый рекорд скрипту task.py. Этот скрипт сообщает серверу Everest, что нужно установить новое значение переменной рекорда, если передаваемое значение меньше текущего. Для этой цели в Everest был реализован специальный протокол. Если значение действительно оказалось меньше текущего, сервер Everest транслирует новое значение всем выполняющимся в данный момент задачам. Они, в свою очередь, передают новое значение адаптерам, а те — решателям.

3. Сведения о задаче коммивояжера

Рассмотрим граф с множеством вершин $V = 1 : n$ (занумерованных от 1 до n). Пусть каждая пара вершин (i, j) соединена ребром длины d_{ij}. Требуется найти гамильтонов путь с наименьшей суммарной длиной ребер. Для каждой пары вершин (i, j) введем булевую переменную x_{ij}, которая принимает значение 1, если ребро (i, j) входит в кратчайший маршрут, и 0 — если нет. Тогда задача коммивояжера может быть записана в виде формул 1–6. Вспомогательные «непрерывные» переменные y_{ij} имеют физический смысл потока, выходящего из первой вершины графа, протекающего через все вершины в порядке прохождения выбранного пути, причем после каждой пройденной вершины величина потока уменьшается на единицу (это давно известный прием сведения задачи коммивояжера к ЛП, например, его можно найти в учебном пособии [**10**]).

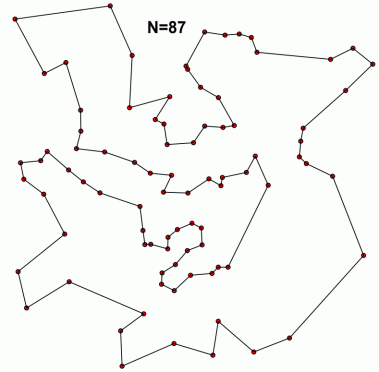

Рис. 4. Пример оптимального маршрута коммивояжера для 87 вершин («городов»)

$$\sum_{i>j} d_{ij}x_{ij} \to \min_{x_{ij},y_{ij}} \quad s.t.: \tag{1}$$

$$\sum_{j\in V, i>j} x_{ij} + \sum_{j\in V, i<j} x_{ji} = 2 \quad \Big(i \in V = \{1:n\}\Big); \tag{2}$$

$$y_{ij} \le \begin{cases} nx_{ij}, & \text{если } i = 1, \\ (n-1)x_{ij}, & \text{если } 1<i<j, \\ (n-1)x_{ji}, & \text{если } j<i \end{cases} \quad \Big((i,j) \in V \times V\Big); \tag{3}$$

$$\sum_{j:(i,j)\in V\times V} y_{ij} - \sum_{j:(i,j)\in V\times V} y_{ji} \le \begin{cases} n-1, & \text{если } i = 1, \\ -1, & \text{если } i>1 \end{cases} \quad \Big(i \in V\Big); \tag{4}$$

$$\sum_{j:(i,j)\in V\times V} y_{ij} \ge 1 \quad \Big(i \in V\Big); \tag{5}$$

$$x_{ij} = \{0,1\}. \tag{6}$$

Сделаем ряд замечаний. Задача коммивояжера интерпретируется как поиск кратчайшего гамильтонова пути обхода точек (городов) на плоскости с координатами (X_i, Y_j). Значение d_{ij} вычисляется как расстояние между точками на плоскости. Для экспериментов значения

Таблица 1. Сравнение времени работы SCIP и крупноблочного алгоритма

N	K	Число подзадач	T(SCIP), мин.	T(dSCIP), мин.
80	4	16	21	2.2
90	5	32	4	2.2
100	6	64	7.7	3.4
110	7	128	>2600	117

(X_i, Y_j) создаются генератором случайных чисел AMPL. После нахождения оптимального маршрута в AMPL результат можно отобразить графически (в SVG-формате) (рис. 4).

Было замечено, что в задаче коммивояжера, для заметного ускорения поиска решения разбиение следует начинать с наименьших расстояний между городами. А именно, числа d_{ij} упорядочивались в порядке возрастания их значений: $d_{i_1j_1} \le d_{i_2j_2} \le d_{i_3j_3} \le d_{i_4j_4} \le d_{i_5j_5} \le ...$

После этого, для фиксации булевых значений, выбирались первые K элементов в указанной последовательности. В проведенных экспериментах:

(1) $K = 0$ (исходная задача);
(2) $K = 1$ (две подзадачи для $N = 70, 80, 90$);
(3) $K = 2$ (четыре подзадачи для $N = 70, 80, 90$);
(4) $K = 3$ (8 подзадач для $N = 70, 80, 90$);
(5) $K = 4$ (16 подзадач для $N = 70, 80, 90$);
(6) $K = 5$ (32 подзадачи для $N = 70, 80, 90$);
(7) $K = 6$ (64 подзадачи для $N = 70, 80, 90, 100$);
(8) $K = 7$ (128 подзадач для $N = 110$).

Подзадачи получаются из исходной после фиксаций значений булевых переменных $x_{i_1j_1}, x_{i_2j_2}, x_{i_3j_3}, ..., x_{i_Kj_K}$ нулем или единицей.

4. Вычислительные эксперименты

Вычислительные эксперименты проводились на трех вычислительных узлах: 4 потока на 2 x Intel Xeon E5620@2.40GHz и 2 x 6 потоков на Intel Xeon E5-2620@2.00GHz. Всего было доступно 16 потоков.

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 SCIP 3.2.1 без какого-либо распараллеливания (время T(SCIP) в таблице 1).

Таблица 2. Зависимость времени решения в секундах от количества зафиксированных переменных (подзадач) при $N = 70$

Задача\K	0 (1)	1 (2)	2 (4)	3 (8)	4 (16)	5 (32)	6 (64)
1	28	31	31	36	27	42	52
2	134	121	106	186	141	192	216
3	122	101	46	52	36	47	87
4	296	116	71	47	101	52	77
5	17	31	21	26	31	43	57
6	256	291	21	251	206	47	322
Сумма:	852	691	**296**	598	542	423	811

Таблица 3. Зависимость времени решения в секундах от количества зафиксированных переменных (подзадач) при $N = 80$

Задача\K	0 (1)	1 (2)	2 (4)	3 (8)	4 (16)	5 (32)	6 (64)
1	336	246	636	391	181	261	142
2	89	36	136	32	92	47	73
3	465	91	156	121	146	222	307
4	2441	2157	461	421	721	908	277
5	226	291	171	171	321	162	227
6	391	1866	4146	571	867	962	2708
Сумма:	3948	4687	5706	**1707**	2328	2562	3734

Далее было проведено по одному «распределенному» запуску на указанных выше вычислительных ресурсах, результаты перечислены в столбце T(dSCIP) таблицы 1. Фиксировались переменные, соответствующие ребрам наименьшей длины в графе расстояний. Для генерации подзадач использовался транслятор AMPL.

Далее была проведена еще одна серия экспериментов (таблицы 2–4). Для нескольких значений N было случайным образом создано по шесть задач (строки таблиц). Каждая из этих задач была разбита на подзадачи шестью различными способами: брались значения K от 1 до 6 (столбцы таблиц). Для каждого полученного набора подзадач запускался крупноблочный алгоритм и время решения записывалось. Время решения исходной задачи записывалось в первый столбец

Таблица 4. Зависимость времени решения (сек.) от числа зафиксированных переменных при $N = 90$

Задача\K	0 (1)	1 (2)	2 (4)	3 (8)	4 (16)	5 (32)	6 (64)
1	73	251	376	206	162	248	528
2	3247	2306	2537	2537	2338	5645	2404
3	160	281	446	346	67	202	79
4	1169	831	1597	766	457	482	1453
5	319	201	791	421	1163	1908	693
6	922	1776	6383	906	7041	18771	10679
Сумма:	5890	5646	12130	**5182**	11228	27256	15836

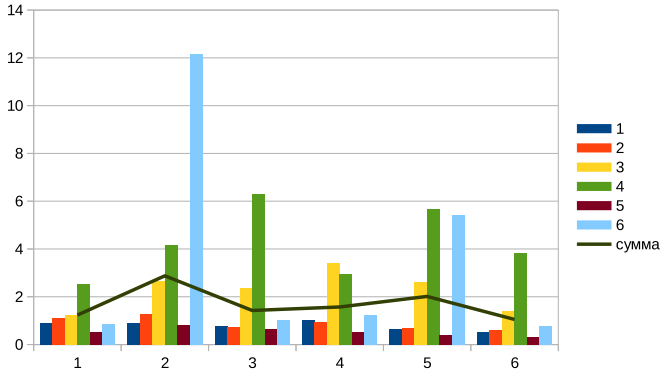

Рис. 5. Зависимость величины ускорения от числа зафиксированных переменных для $N = 70$

таблиц. Последней строкой в таблицах приведены суммы времен решения для каждого значения K. Следует отметить, что во всех случаях использовались настройки МВГ-решателей «по умолчанию».

Данная серия экспериментов проводилась на расширенном наборе вычислительных ресурсов из четырех хостов и 26 потоков:

(*1*) Хост irbis1: 8 потоков 2 x Intel Xeon E5620 @ 2.40GHz,
(*2*) Хост fuji: 6 потоков Intel Xeon E5410 @ 2.33GHz,
(*3*) Хост restopt1: 6 потоков Intel Xeon E5-2620 @ 2.00GHz,
(*4*) Хост restopt2: 6 потоков Intel Xeon E5-2620 @ 2.00GHz.

Результаты данного эксперимента также представлены на рис. 5–7. На них изображена зависимость ускорения (времени решения исходной

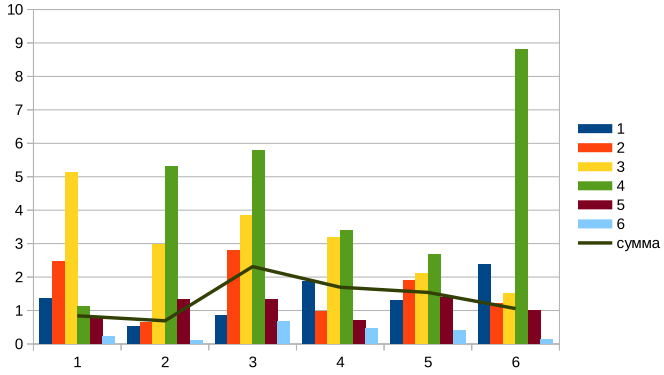

Рис. 6. Зависимость величины ускорения от числа зафиксированных переменных для $N = 80$

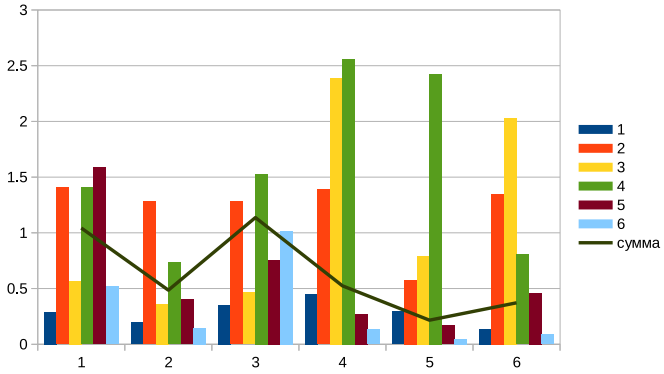

Рис. 7. Зависимость величины ускорения от числа зафиксированных переменных для $N = 90$

задачи одним решателем, деленного на время решения в распределенном режиме) от числа зафиксированных переменных K. Значение ускорения отложено по вертикальной оси, а число зафиксированных переменных — по горизонтальной. Для отдельных экземпляров тестовых задач используются столбцы, а для суммарного ускорения — линия.

На графиках видно, что зависимость времени решения от числа подзадач ведет себя довольно непредсказуемо. Например, при зна-

Таблица 5. Потраченное машинное время для шестой задачи с $N = 90$

K	restopt1	restopt2	irbis1	fuji
1	0	1560 (1)	1775 (1)	0
2	3142 (1)	3180 (1)	3340 (1)	6382 (1)
3	1231 (2)	1140 (2)	1810 (2)	435 (2)
4	13254 (4)	13972 (4)	4630 (4)	6698 (4)
5	13817 (6)	49032 (6)	45106 (8)	26652 (6)
6	6087 (6)	31200 (6)	29778 (8)	14603 (6)

чении $K = 4$ все подзадачи начинают решаться одновременно и нет задач, ожидающих своей очереди на выполнение. При этом число выделенных на решение задачи ресурсов в два раза больше, чем при $K = 3$. Однако, ускорение при $K = 4$ часто оказывается ниже ускорения при $K = 3$.

По графикам видно, что, если судить по суммарному времени решения (фактически — по усредненному показателю), то максимальное ускорение наблюдалось при фиксации 2 или 3 переменных. Стоит отметить, что при $K \geq 5$ подключенных решателей не хватает, чтобы запустить все подзадачи сразу. Это причина «замедления» при $K \geq 5$.

В таблице 5 представлено машинное время, потраченное на вычислительных ресурсах при решении шестой задачи для $N = 90$ (в скобках указано число одновременно выполняемых подзадач). Из этой таблицы можно видеть, что Everest распределяет задания довольно равномерно по вычислительным ресурсам. Например, четыре задания для задачи с $K = 2$ были запущены каждая на отдельном хосте. Хотя свободных ресурсов было достаточно, чтобы запустить все четыре задания на одном хосте.

5. Заключение

В статье описывается программная реализация параллельной крупнозернистой схемы метода ветвей и границ для решения задач дискретной оптимизации в среде веб-сервисов на основе платформы Everest. Для обмена рекордными значениями целевой функции между параллельно работающими решателями, реализующими алгоритм МВГ, используется подсистема обмена сообщениями в среде приложений Everest.

Для задачи коммивояжера различных размерностей представленная в работе схема распараллеливания метода ветвей и границ продемонстрировала заметное ускорение по сравнению со временем решения тех же задач однопоточным экземпляром МВГ-решателя. Для разбиения исходной задачи на подзадачи применялся эвристический метод, реализованный на языке оптимизационного моделирования AMPL. Исходные коды разработанной системы доступны по адресу https://github.com/ssmir/dcbc.

Список литературы

[1] Л. Д. Попов. «Опыт многоуровневого распараллеливания метода ветвей и границ в задачах дискретной оптимизации», *Автомат. и телемех.*, 2007, №5, с. 171–181, URL: http://www.mathnet.ru/php/getFT.phtml?jrnid=at&paperid=993&what=fullt&option_lang=rus ↑ 106

[2] E. P. Mancini, S. Marcarelli, I. Vasilyev, U. Villano. "A grid-aware MIP solver: Implementation and case studies", *Future Generation Computer Systems*, **24**:2 (2008), pp. 133–141. ↑ 106

[3] M. R. Bussieck, M. C. Ferris, A. Meeraus. "Grid-enabled optimization with GAMS", *INFORMS Journal on Computing*, **21**:3 (2009), pp. 349–362. ↑ 106

[4] С. А. Смирнов, В. В. Волошинов. «Эффективное применение пакетов дискретной оптимизации в облачной инфраструктуре на основе эвристической декомпозиции исходной задачи в системе оптимизационного моделирования AMPL», *Программные системы: теория и приложения*, **7**:1 (2016), с. 29–46, URL: http://psta.psiras.ru/read/psta2016_1_29-46.pdf ↑ 106

[5] С. А. Смирнов, В. В. Волошинов. «Предварительная декомпозиция задач дискретной оптимизации для ускорения алгоритма ветвей и границ в распределенной вычислительной среде», *Компьютерные исследования и моделирование*, **7**:3 (2015), с. 719–725, URL: http://crm.ics.org.ru/uploads/crmissues/crm_2015_3/15746.pdf ↑ 106

[6] O. Sukhoroslov, S. Volkov, A. Afanasiev. "A Web-Based Platform for Publication and Distributed Execution of Computing Applications", *Proc. 14th International Symposium on Parallel and Distributed Computing*, ISPDC 2015 (Limassol, Cyprus, 29 June–01 July. 2015), IEEE, 2015, pp. 175–184. ↑ 107

[7] G. Gamrath, T. Fischer, T. Gally, A. M. Gleixner, G. Hendel, T. Koch, S. J. Maher, M. Miltenberger, et al. *The SCIP Optimization Suite 3.2*, ZIB Report, 2016, 15–60. ↑ 107

[8] H. Mittelmann. *Mixed Integer Linear Programming Benchmark*, 2016, URL: http://plato.asu.edu/ftp/milpc.html ↑ 107

[9] R. Fourer, D. M. Gay, B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*, second edition, Duxbury Press/Brooks/Cole Publishing Company, 2002, 538 p. ↑ 108

[10] Е. В. Алексеева, *Построение математических моделей целочисленного линейного программирования. Примеры и задачи*, Учеб. пособие, НГУ, Новосибирск, 2012, 132 с. ↑ 109

Пример ссылки на эту публикацию:

В. В. Волошинов, С. А. Смирнов. «Оценка производительности крупноблочного алгоритма метода ветвей и границ в вычислительной среде Everest», *Программные системы: теория и приложения*, 2017, **8**:1(32), с. 105–119.

URL: http://psta.psiras.ru/read/psta2017_1_105-119.pdf

Об авторах:

Владимир Владимирович Волошинов

Зав. лаб. Ц-3 ИППИ РАН, старший научный сотрудник, к.ф.-м.н. Научные интересы: теория и практика применения оптимизационных моделей в разных областях науки и техники; различные вопросы распределенных вычислений.

e-mail: vladimir.voloshinov@gmail.com

Сергей Андреевич Смирнов

И.о. научного сотрудника лаборатории № Ц-1 ИППИ РАН, к.т.н. Область научных интересов включает распределенные вычисления, облачные вычисления, а также использование пакетов оптимизации.

e-mail: sasmir@gmail.com

Vladimir Voloshinov, Sergey Smirnov. *Evaluation of a Coarse-Grained Branch-and-Bound Algorithm in the Everest Computing Environment.*

Abstract. The study concerned the coarse-grained approach to the implementation of the parallel branch-and-bound (B&B) method. Feasible domain of the mixed integer programming problem is divided into several subsets by fixing some of the integer variables. It corresponds to decomposition of the initial problem into subproblems, which are being solved in parallel by a pool of solvers. When solver finds an incumbent value it is broadcasted to other solvers. Such exchange of optimal objective value's bounds allows to speed up the solution of subproblems via reducing number of nodes of B&B search tree, which are to be explored. Solver life cycle and incumbent values' exchange are managed by the Everest Web-based platform for distributed computing, everest.distcomp.org. The system has been tested on several randomly generated mixed-integer programming problems and a noticeable speedup has been detected. (*in Russian*).

Key words and phrases: discrete optimization, branch and bound algorithm, coarse-grained parallelism.

References

[1] L. D. Popov. "Experience of multilevel parallelizing of the branch and bound method in discrete optimization problems", *Automation and Remote Control*, **68**:5 (2007), pp. 901–911.

[2] E. P. Mancini, S. Marcarelli, I. Vasilyev, U. Villano. "A grid-aware MIP solver: Implementation and case studies", *Future Generation Computer Systems*, **24**:2 (2008), pp. 133–141.

[3] M. R. Bussieck, M. C. Ferris, A. Meeraus. "Grid-enabled optimization with GAMS", *INFORMS Journal on Computing*, **21**:3 (2009), pp. 349–362.

[4] S. A. Smirnov, V. V. Voloshinov. "Effective use of discrete optimization solvers in cloud infrastructure on the basis of heuristic decomposition of the initial problem by optimization modeling system AMPL", *Programmnyye sistemy: teoriya i prilozheniya*, **7**:1 (2016), pp. 29–46 (in Russian), URL: http://psta.psiras.ru/read/psta2016_1_29-46.pdf

[5] S. A. Smirnov, V. V. Voloshinov. "Pre-decomposition of discrete optimization problems to speed up the branch and bound method in a distributed computing environment", *Komp'yuternyye issledovaniya i modelirovaniye*, **7**:3 (2015), pp. 719–725 (in Russian), URL: http://crm.ics.org.ru/uploads/crmissues/crm_2015_3/15746.pdf

[6] O. Sukhoroslov, S. Volkov, A. Afanasiev. "A Web-Based Platform for Publication and Distributed Execution of Computing Applications", *Proc. 14th International Symposium on Parallel and Distributed Computing*, ISPDC 2015 (Limassol, Cyprus, 29 June–01 July. 2015), IEEE, 2015, pp. 175–184.

[7] G. Gamrath, T. Fischer, T. Gally, A. M. Gleixner, G. Hendel, T. Koch, S. J. Maher, M. Miltenberger, et al. *The SCIP Optimization Suite 3.2*, ZIB Report, 2016, 15–60.

[8] H. Mittelmann. *Mixed Integer Linear Programming Benchmark*, 2016, URL: http://plato.asu.edu/ftp/milpc.html

[9] R. Fourer, D. M. Gay, B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*, second edition., Duxbury Press/Brooks/Cole Publishing Company, 2002, 538 p.

[10] Ye. V. Alekseyeva, *Construction of mathematical models of integer linear programming. Examples and problems*, Ucheb. posobiye, NGU, Novosibirsk, 2012 (in Russian), 132 p.

Sample citation of this publication:

Vladimir Voloshinov, Sergey Smirnov. "Evaluation of a Coarse-Grained Branch-and-Bound Algorithm in the Everest Computing Environment", *Program systems: Theory and applications*, 2017, **8**:1(32), pp. 105–119. (*In Russian*).

URL: http://psta.psiras.ru/read/psta2017_1_105-119.pdf