ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №1(28), 2016, с. 29–46

УДК 004.75

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

Аннотация. Пусть процесс поиска решения в некоторой задаче дискретной оптимизации пакетом, реализующим алгоритм ветвей и границ, занимает определенное время. Можно ли ускорить решение той же задачи если нам доступна вычислительная среда, где можно запустить несколько одновременно работающих «экземпляров» того же пакета оптимизации? В докладе рассматривается способ получить заметное ускорение для пакетов с открытым кодом в виртуальной многопроцессорной вычислительной среде. В основе подхода:

(1) предварительная декомпозиция исходной задачи на несколько подзадач путем фиксации целочисленных (булевых) значений части дискретных переменных, выбираемых согласно некоторому эвристическому правилу, реализованному в форме программы на высокоуровневом языке оптимизационного моделирования AMPL;

(2) одновременное решение полученных подзадач экземплярами того же пакета, с добавленной возможностью обмена найденными рекордными значениями целевой функции.

Предлагаемый подход привлекает относительной простотой программной реализации и демонстрируется на примерах решения задачи коммивояжера и составления расписаний назначения работ исполнителям.

Ключевые слова и фразы: дискретная оптимизация, метод ветвей и границ, предварительная эвристическая декомпозиция.

Введение

Для решения трудоемких задач дискретной оптимизации широкое распространение получили комбинаторные методы, среди которых наиболее часто встречаются методы ветвей и границ (МВГ).

Поддержано грантом РФФИ 13-07-00987.

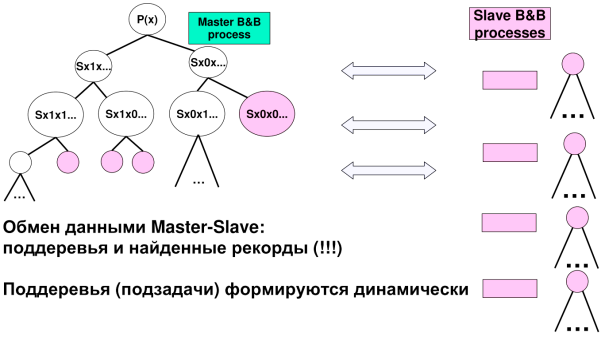

Рис. 1. Принцип реализации «мелкозернистой» схемы МВГ в РВС

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, динамически формируемого в ходе работы алгоритма некоторым координирующим процессом (процессором). Каждому узлу (n) дерева соответствует:

(1) некоторая дискретная подзадача S(n), полученная из исходной в результате фиксации значений части целочисленных переменных;
(2) вспомогательная «непрерывная» (оценочная) подзадача B(n), полученная из подзадачи S(n) ослаблением условий целочисленности.

Работа алгоритма может существенно зависеть от порядка выбора узлов дерева (правил его обхода). Традиционно ускорение работы МВГ достигается за счет применения механизма параллельных вычислений. Обычно речь идет о сочетании двух приемов:

(1) параллельная обработка очереди оценочных подзадач некоторым пулом солверов;
(2) выделение дискретных подзадач S(n) для отдельной обработки свободным координирующим процессом (рис. 1).

Программная реализация такого «мелкозернистого» подхода обычно осуществляется на основе низкоуровневых программных технологий параллельных вычислений (MPI, OpenMP) и требует значительных трудозатрат квалифицированных программистов.

Существует также крупноблочная (или крупнозернистая, или конкурентная) схема распараллеливания метода ветвей и границ [1].

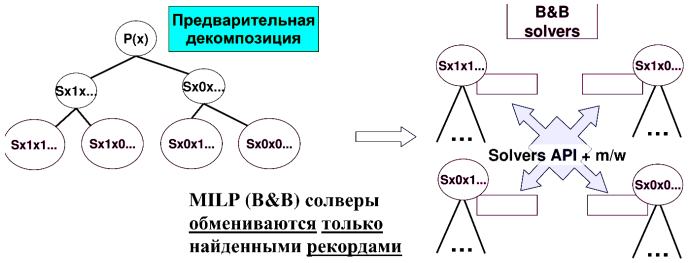

Рис. 2. Принцип реализации «крупноблочной» схемы МВГ в РВС

В ее основе — параллельное решение предварительно сформированного набора подзадач с обменом информацией о найденных в ходе решения подзадач значениях целевой функции на допустимых решениях (т.н. рекордах). 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 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 [1–3], но широкого применения пока не получил.

В данной работе представлена реализация крупноблочной схемы распараллеливания метода ветвей и границ для частично-целочисленных задач оптимизации (рис. 2). Данная реализация обеспечивает запуск подзадач на некотором предварительно заданном множестве хостов и обмен рекордами между процессами, решающими подзадачи. В основе реализации лежит использование существующих пакетов оптимизации с открытым исходным кодом, а именно CBC и SCIP. Представленная система требует предварительного разбиения исходной задачи на подзадачи. Пользователь может самостоятельно, ос-

новываясь на собственных знаниях о задаче, реализовать алгоритм разбиения, например, средствами языка оптимизационного моделирования AMPL.

Указанная система была апробирована на двух задачах:

- классической задаче коммивояжера;
- о составлении расписания для минимизации времени обработки очереди заданий разной трудоемкости пулом исполнителей различной производительности.

Обе указанные задачи были сформулированы в виде задач линейного программирования с частично-булевыми переменными (MILP). («Рецепты» сведения различных задач дискретной оптимизации к задачам MILP можно найти в учебном пособии [4]).

1. 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 (1) при условиях (2)–(6):

$$\sum_{i>j} d_{ij}x_{ij} \to \min_{x_{ij},f_{ij}}, \tag{1}$$

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

$$f_{ij} \le \left\{ \begin{array}{ll} n, & i = 1 \\ n-1, & i > 1 \end{array} \right\} \cdot \left\{ \begin{array}{ll} x_{ij}, & i < j \\ x_{ji}, & i > j \end{array} \right\}, \quad (i, j) \in V \times V; \tag{3}$$

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

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

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

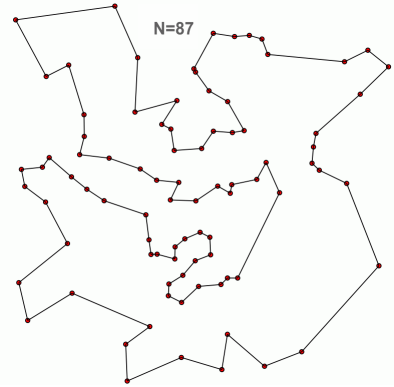

Рис. 3. Пример оптимального маршрута коммивояжера для 87 вершин («городов»)

Вспомогательные «непрерывные» переменные f_{ij} имеют физический смысл потока, выходящего из первой вершины полного графа, протекающего через все вершины, причем в каждой вершине величина входящего потока на единицу больше, чем величина вытекающего (это давно известный прием сведения задачи коммивояжера к ЛП).

Сделаем ряд замечаний. Задача коммивояжера интерпретируется как поиск кратчайшего гамильтонового 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 (X_i, Y_j) создаются генератором случайных чисел AMPL. После нахождения оптимального маршрута в AMPL результат можно визуализировать в формате графического файла в SVG-формате (рис. 3).

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: $d_{i_1 j_1} \le d_{i_2 j_2} \le d_{i_3 j_3} \le d_{i_4 j_4} \le d_{i_5 j_5} \le \ldots$. После этого, для фиксации булевых значений, выбирались первые K элементов в указанной последовательности. В проведенных экспериментах:

(1) K=2 (четыре подзадачи для N=90);
(2) K=3 (8 подзадач для N=90, 100);
(3) K=4 (16 подзадач для N=80, 90);
(4) K=5 (32 подзадач для N=90);

(5) K=6 (64 подзадачи для N=100);
(6) K=7 (128 подзадач для N=110).

Указанные подзадачи получаются из исходной после фиксаций булевых переменных $x_{i_1 j_1}, x_{i_2 j_2}, x_{i_3 j_3}, x_{i_4 j_4}, x_{i_5 j_5}, ..., x_{i_K j_K}$ значениями 0 или 1.

2. Сведения о задаче составления расписания (TWS, Task-Worker-Scheduling)

Рассматривается задача обработки очереди (вычислительных) заданий набором исполнителей (исполнительных устройств). Ограничимся полностью детерминированной постановкой:

- заранее известны трудоемкость всех заданий и время их постановки в очередь;
- производительность исполнителей известна и остается неизменной.

Очередь заданий представлена последовательностью моментов времени их поступления в очередь планировщика $\{T_K, k = 1 : K\}$, где k — номер задания (задачи), причем $T_K \leq T_{k+1}$, т.е. нумерация заданий соответствует порядку их постановки в очередь. В условиях полной определенности каждое задание характеризуется своей трудоемкостью τ_k, имеющей размерность времени и равной продолжительности решения задачи некоторым эталонным исполнителем единичной производительности. С учетом введенных обозначений, очередь заданий можно моделировать последовательностью $J = \{j_k, k = 1 : K\}$, где $j_k = (\tau_k, T_k)$.

Множество (пул) исполнителей будем обозначать конечным множеством $W = \{w_n, n = 1 : N\}$. Для простоты будем предполагать, что каждый исполнитель в каждый момент времени может выполнять одно задание из очереди J. Кроме того, будем предполагать, что выполнение задания требует непрерывной работы исполнителя, т.е. задание не может быть приостановлено, чтобы исполнитель начал обрабатывать другую задачу, и затем продолжено с «точки останова». Прерванное исполнение нужно будет начать заново.

В условиях полной определенности каждый исполнитель имеет «безразмерную» производительность p_k, равную отношению продолжительности решения им некоторой эталонной задачи (единичной трудоемкости) ко времени решения той же задачи эталонным исполнителем (единичной производительности). В этих обозначениях

продолжительность выполнения k-го задания n-ым исполнителем находится по формуле $d_{kn} = \frac{\tau_k}{p_n}$.

Условия обработки очереди заданий с учетом времени их появления формулируются в виде системы линейных неравенств с частично-целочисленными переменными. Приведем только систему ограничений, на которую затем можно «наложить» различные критерии (метрики) качества работы алгоритма.

Любое расписание обработки очереди заданий может быть однозначно задано набором переменных $t_k \in R^1, x_{kn} = \{0,1\}$, $k = 1 : K$, $n = 1 : N$, где t_k — непрерывная переменная, равная времени начала выполнения k-го задания; x_{kn} — булевая переменная, принимающая значения $\{0,1\}$ (равная 1, если k-е задание выполняется n-ым исполнителем, и 0 — если другим).

Перечислим линейные неравенства, определяющие допустимые значения переменных $\{t_k, x_{kn}\}$, соответствующих реализуемому расписанию:

(1) $t_k \geq T_k$, $k = 1 : K$ — задание может начать выполняться только после появления в очереди;

(2) $\sum_{n=1:N} x_{kn} \geq 1$, $k = 1 : K$ — условие выполнения задания хотя бы одним исполнителем;

(3) $t_k + x_{kn}\frac{\tau_k}{p_n} \leq t_{k'} + (2 - x_{kn} - x_{k'n}) \cdot C$, $k, k' = 1 : K$, $k < k'$, $n = 1 : N$, где C — достаточно большая константа, зависящая только от параметров J и W (как ее определить — см. далее) — обеспечивает «непересечение» интервалов времени выполнения заданий, назначенных одному исполнителю при условии, что очередность выполнения заданий (этим одним исполнителем) соответствует порядку их поступления в «основную» очередь (задания с большим номером выполняются позже). С одной стороны, это требование кажется естественным, с другой — от него можно отказаться за счет дополнительных булевых переменных.

Покажем, что для любой метрики оценки качества алгоритма, связанной с «уменьшением» времени обработки очереди заданий, значение C можно выбрать достаточно большим, чтобы указанные неравенства были бы «несущественными», т.е. тривиально выполнялись для любого «разумного» расписания. Формальная формулировка звучит так: при поиске оптимальных расписаний можно ограничиться такими значениями переменных t_k, x_{kn}, для которых неравенство

$$t_k + \frac{\tau_k}{p_n} x_{kn} \leq t_{k'} + C$$

будет заведомо выполнено.

Заметим, что C можно выбрать равным верхней оценке продолжительности выполнения всех заданий одним исполнителем. Ясно, что

$$p_{\min} = \min\{p_n,\ n{=}1{:}N\}.$$

Также введем обозначение для самого трудного задания:

$$\tau_{\max} = \max\{\tau_k,\ k{=}1{:}K\}.$$

Тогда требуемое значение C можно получить в результате следующей рекуррентной процедуры, состоящей из трех этапов:

$$C_1 := 0; \tag{7}$$

$$C_k := \max\{C_{k-1} + \frac{\tau_{k-1}}{p_{\min}}, T_k\}, \quad k = 2 : K; \tag{8}$$

$$C := C_K + \frac{\tau_{\max}}{p_{\min}}. \tag{9}$$

Заметим, что справа от оператора := в среднем «цикле» стоит минимально возможное время начала выполнения k-го задания самым медленным исполнителем.

Следует отметить, что похожие приемы описания расписаний выполнения работ в форме системы линейных соотношений с булевыми переменными уже применялись в работах по исследованию операций. В свою очередь, сами по себе они являются частными случаями т.н. дихотомических ограничений, которые упоминаются и в более ранних монографиях по дискретному программированию.

Если использовать введенные выше обозначения и формулы, то в условиях полной определенности минимально возможное время выполнения всех заданий может быть получено в результате решения следующей задачи линейного программирования (10) с условиями (11)–(15) и булевыми переменными (значение константы C определяется по формулам (7)–(9)):

$$z \to \min_{t_k, x_{kn}, z}, \tag{10}$$

$$t_k + \frac{\tau_k}{p_n} x_{kn} \le z, \quad k = 1 : K, n = 1 : N; \tag{11}$$

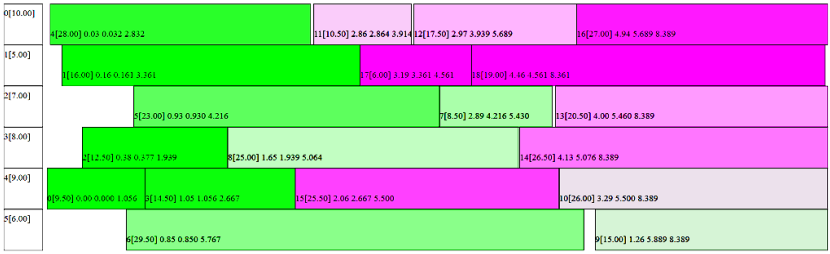

Рис. 4. Пример оптимального расписания 19 заданий для 6 исполнителей

$$t_k \geq T_k, \quad k = 1 : K; \tag{12}$$

$$\sum_{n=1:N} x_{kn} = 1, \quad k = 1 : K; \tag{13}$$

$$t_k + \frac{\tau_k}{p_n} x_{kn} \leq t_{k'} + C \times (2 - x_{kn} - x_{k'n}), \tag{14}$$

$$k = 1 : K, k < k' \leq K, n = 1 : N;$$

$$z, t_k \in R^1, x_{kn} = 0, 1, k = 1 : K, n = 1 : N. \tag{15}$$

В ходе экспериментов случайным образом варьировалось время постановки заданий в очередь. После нахождения оптимального расписания результат можно визуализировать в формате графического файла в SVG-формате (см. рис. 4). На этом рисунке приведены характерные значения трудоемкостей заданий (в квадратных скобках, после номера задания в «цветных» прямоугольниках) и производительность исполнителей (в квадратных скобках в первой колонке «серых» прямоугольниках).

Было замечено, что в задаче TWS, для заметного ускорения поиска решения, разбиение следует начинать с переменных x_{kw}, соответствующих наименьшим интервалам (продолжительностей) выполнения. А именно, числа $d_{k,n} = \frac{\tau_k}{p_n}$ упорядочивались в порядке возрастания их значений:

$$d_{i_1,j_1} \leq d_{i_2,j_2} \leq d_{i_3,j_3} \leq d_{i_4,j_4} \leq d_{i_5,j_5} \leq \ldots.$$

После этого выбирались первые K_{fix} элементов в указанной последовательности, и фиксировались значения соответствующих булевых переменных. В проведенных экспериментах $K_{fix} = 6$ (64 подзадачи) для $N = 19$, $W = 6$. Указанные подзадачи получаются из исходной после фиксаций булевых переменных x_{i_1,j_1}, x_{i_2,j_2}, x_{i_3,j_3}, x_{i_4,j_4}, $x_{i_5,j_5}, \ldots, x_{i_K,j_K}$ значениями 0 и 1. Как и для TSP задачи, выбор K_{fix} переменных порождает $2^{K_{fix}}$ подзадач.

3. Вычислительные эксперименты

Для обоих указанных выше типов задач было получено более чем двукратное ускорение. С учетом малой стоимости аренды одного ядра в коммерческих облачных инфраструктурах, часто может быть целесообразно использование большого числа вычислителей даже с двукратным ускорением. Это делает не столь значимым общепризнанный критерий эффективности ускорения, учитывающий число процессоров, занятых параллельным алгоритмом.

Задача коммивояжера

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). Также в перечисленных ниже запусках подзадачи распределялись по вычислительным узлам предельно просто: сначала полностью загружался первый узел (8 потоков), а затем второй (4 потока). Освободившийся поток получал очередное задание из очереди. После исчерпания очереди подзадач освободившиеся потоки простаивали.

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

Вначале каждая из исходных задач была решена без распараллеливания двумя пакетами (каждый работал в однопоточном процессе): CBC (время T(CBC) в таблице 1); SCIP (время T(SCIP) 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.

Таблица 1. Решение в однопоточном процессе, мин.

N	T(CBC)	T(SCIP)
80	5,3	1,6
90	20,3	6,0
100	624	1,0
110	>10000	75

Таблица 2. Продолжительность распределенного решения, мин.

N	фиксировано x_{ij}	подзадач	T(CBC)	T(dCBC)
80	4	16	5,3	1,9
90	5	32	20,3	11
100	6	64	624	229
110	7	128	>10000	1212

Далее было проведено по одному «распределенному» запуску на указанных выше вычислительных ресурсах, результаты сведены в таблицу 2. Фиксировались переменные, соответствующие ребрам наименьшей длины в графе расстояний между городами. Деление на подзадачи проводилось встроенными средствами языка AMPL. Как видим, в среднем, было получено более чем двукратное ускорение.

Также делались однократные «распределенные» запуски, где размер исходной задачи фиксировался, а варьировалось число фиксируемых переменных. Деление на подзадачи производилось средствами библиотеки солверов AMPL. Результаты представлены в таблице 3. Видно, что время решения в распределенном режиме ощутимо зависит от числа подзадач, на которые поделена исходная задача. Более того, время решения начинает резко возрастать если число подзадач превышает число доступных вычислительных ядер.

Задача составления расписания

Первый вычислительный эксперимент проводился для системы dSCIP, установленной на вычислительной инфраструктуре, представленной в таблице 4. Здесь первые две машины — «реальные» серверы с многоядерными процессорами, а последние две — виртуальные (с указанными характеристиками), управляемые при помощи Xen-Manager.

Таблица 3. Время решения в мин. при $N = 90$ и $N = 100$

Фикс. x_{ij}	Подзадач	T(dCBC)
0	1	20,3
2	4	16,0
3	8	4,4
4	16	8,5
5	32	8,6

Фикс. x_{ij}	Подзадач	T(dCBC)
0	1	624
3	8	202
6	64	1627

Таким образом, система dSCIP имела в своем распоряжении 40 экземпляров одновременно запущенных процессов SCIP. В последней колонке указано время решения тестовой задачи одним экземпляром пакета SCIP. Это значение дает оценку производительности используемых ресурсов.

Все машины работали в одной локальной сети. Объем информации, пересылаемой в системе dCBC/SCIP, невелик: необходимо передавать подзадачи исполнителям, а также несколько раз разослать одно числовое значение очередного, найденного одним из процессов, рекорда целевой функции, что происходит сравнительно редко. Например, работая на инфраструктуре, указанной в таблице 4, в ходе обработки 64 задач было разослано 40 рекордных значений целевой функции. Для понимания принципа работы метода МВГ полезно отметить, что рекорд, который оказался решением, был найден примерно на 490-й секунде работы. Оставшееся время алгоритм «доказывал», что найденное допустимое решение оптимально.

Решение той же задачи, но представленной набором из 64 подзадач, подготовленных программой на языке AMPL, заняло 12 минут. Необходимо отметить значительную неоднородность вычислительной среды, указанной в таблице 4: производительность многопроцессорных узлов различается более, чем вдвое, а на первых двух серверах число потоков таково, что задействуется механизм HyperThreading. Такая неоднородность усложняет оценку ускорения, так как приходится выбирать, с каким из значений из третьей колонки сравнивать время распределенного запуска. Если сравнить это время с наименьшим значением из третьей колонки таблицы, то мы получаем менее 25% ускорения (при использовании 40 одновременно работающих пакетов SCIP) по сравнению с тем временем, что уходит на решение MILP задачи целиком одним экземпляром алгоритма МВГ. С другой стороны,

Таблица 4. Вычислительная инфраструктура для тестов применения dSCIP к задаче TWS

Номер сервера и параметры процессора	Число работающих процессов SCIP	Продолжительность решения одним SCIP, мин.
1, Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz	8	15
2, Intel(R) Xeon(R) CPU E5620 @ 2.40GHz	16	23
3 (вирт.), Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz	8	37
4 (вирт.), Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz	8	37
	Всего процессов: 40	Время dSCIP: 12

вероятно, лучше сравнивать с некоторой средней производительностью по собранному кластеру.

Чтобы исключить влияние вычислительной неоднородности на возможности исследуемого подхода, следующий эксперимент проводился в полностью виртуальной среде из двух 20-ти процессорных серверов, заказанных в коммерческой облачной инфраструктуре: 2x (20-cores KVM Intel Xeon @ 2.4Ghz). Для проведения эксперимента был создан образ виртуальной машины Linux Ubuntu 12.04 с установленным компонентами dCBC/SCIP. Таким образом, исходная задача, разбитая на 64 подзадачи, решалась 40 экземплярами пакета SCIP. Наглядное представление результатов приведено на рис. 5. Верхней горизонтальной линией, с надписью «исходная задача, 1xSCIP», отмечена продолжительность решения задачи одним экземпляром пакета SCIP (около 25 мин. или 1500 сек.). Нижняя линия «(64 подзадачи => SCIP ...)» показывает, что при параллельной обработке 40 процессами с обменом рекордами в системе dSCIP было получено почти двойное сокращение времени поиска решения (точнее, 12 минут, как и при первом запуске). Заметим, что мы оцениваем не эффективность распараллеливания, а только ускорение (т.е. не учитываем объем привлеченной вычислительной мощности). По-нашему мнению, ввиду все большей дешевизны вычислительных устройств, основное внимание следует уделять их эффективному использованию, с целью добиться заметного повышения производительности.

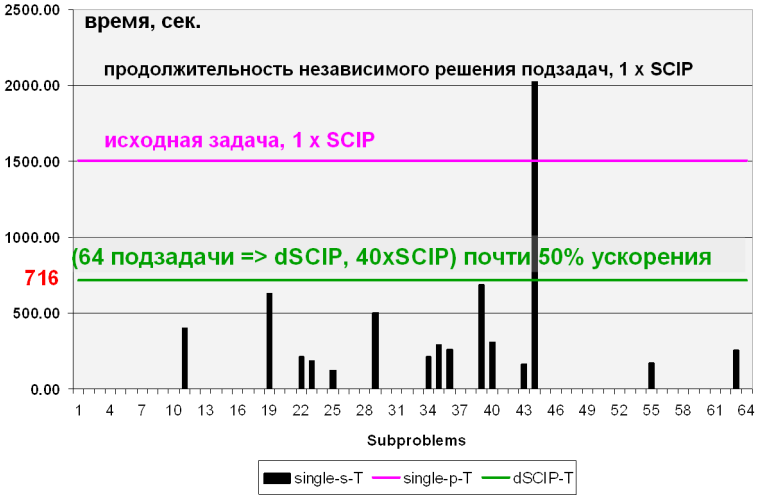

Рис. 5. Результаты dSCIP для задачи TWS в однородной вычислительной среде

На том же рисунке вертикальные столбики обозначают продолжительность независимого решения каждой из 64 подзадач одним из процессоров указанных виртуальных машин без обмена значениями рекордов. Видно, что если бы одновременно работали даже 64 пакета SCIP, то решение всех подзадач без обмена рекордами заняло бы больше времени (чуть более 2000 сек., на 44-й задаче), чем решение исходной «полноразмерной» задачи TWS. Возможно, было бы полезно внести в систему средства равномерного распределения вычислительной мощности по всем созданным подзадачам (например, периодически приостанавливать обработку отдельных подзадач, чтобы возобновить обработку остальных).

4. Заключение

Предложенный подход представляется одним из простейших способов получить заметный эффект ускорения работы алгоритма ветвей и границ в многопроцессорной вычислительной среде. Для его программной реализации нужно лишь реализовать обмен (рассылку и доставку) сообщениями о рекордных значениях целевой функции, обнаруженных параллельно работающими пакетами дискретной оптимизации, между процессами, в которых работают эти пакеты. Для пакетов

с открытым кодом это возможно реализовать на основе известного программного интерфейса пакета и той или иной системы обмена сообщениями.

Как показали предварительные исследования, эффективность предложенного подхода существенно зависит от выбранной схемы предварительной декомпозиции исходной задачи на подзадачи. Именно здесь удобно использовать тот или иной алгебраический язык оптимизационного моделирования (мы использовали AMPL [5]), но аналогичные возможности предоставляют и иные языки, например, GAMS или Mosel [6]). Такие языки (и соответствующие интерпретаторы) применяются как для записи исходной задачи (параметров, переменных, ограничений и целевой функции), так и для эвристического анализа параметров в ходе декомпозиции и генерации подзадач в форме, которую можно передать на вход пакетам. Например, для AMPL и GAMS такой формой являются т.н. «стаб-файлы» (stubs), содержащие всю информацию об «экземпляре» задачи.

Предлагаемый подход имеет естественные границы применения. Например, экспоненциальный рост числа подзадач при увеличении числа «фиксируемых» переменных. Более того, упоминаемые «эвристические правила» декомпозиции предполагают некоторый неформальный анализ исходного класса задач.

Тем не менее, обнаруженное нами, на двух примерах известных задач, заметное ускорение за счет «простого» увеличения числа параллельно работающих пакетов оптимизации выглядит как один из возможных ответов на поставленный в работе [7] вопрос.

Список литературы

[1] Л. Д. Попов. «Опыт многоуровневого распараллеливания метода ветвей и границ в задачах дискретной оптимизации», *Автоматика и телемеханика*, 2007, №5, с. 171–181. ↑ 30,31

[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. ↑ 31

[3] M. R. Bussieck, M. C. Ferris, A. Meeraus. “Grid-enabled optimization with GAMS”, *INFORMS Journal on Computing*, **21**:3 (2009), pp. 349–362. ↑ 31

[4] Е. В. Алексеева, *Построение математических моделей целочисленного линейного программирования. Примеры и задачи*, Учеб. пособие, НГУ, Новосибирск, 2012, 132 с. ↑ 32

[5] 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. ↑ 43

[6] J. Kallrath, *Algebraic Modeling Systems: Modeling and Solving Real World Optimization Problems*, Applied Optimization, vol. **104**, Springer Science & Business Media, 2012, 254 p. ↑ 43

[7] T. Koch, T. Ralphs, Y. Shinano. Could we use a million cores to solve an integer program? *Mathematical Methods of Operations Research*, **76**:1 (2012), pp. 67–93. ↑ 43

Рекомендовал к публикации *Программный комитет*
Четвёртого национального суперкомпьютерного форума ***НСКФ-2015***

Об авторах:

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

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

Пример ссылки на эту публикацию:

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*, 2016, **7**:1(28), с. 29–46.

URL: http://psta.psiras.ru/read/psta2016_1_29-46.pdf

Sergey Smirnov, Vladimir 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.*

Abstract. Let's take some discrete optimization problem and a time interval to find its solution by some branch and bound solver. Can we reduce time of solving if a number of instances of the same solver will run simultaneously with the same problem? We have got significant acceleration with open source solvers in multi-processors cloud computing environment. Our approach is based on the following:

(1) preliminary decomposition of the given problem into sub-problems via fixing integer values of some discrete variables selected in accordance with some heuristic rule implemented as a program in optimization modelling language AMPL;

(2) parallel solving of sub-problems by a pool of solvers which run simultaneously and exchange incumbents (bounds of optimal values in branch-and-bound method) they found in processing of sub-problems.

The approach is attractive due to the relative simplicity of program implementation. It has been verified on "Traveling Salesman (TSP)" and "Task-to-Worker scheduling" problems. (*In Russian*).

Key words and phrases: discrete optimization, branch and bound algorithm, prior heuristic decomposition.

References

[1] L. D. Popov. "Experience of multilevel parallelizing of the branch and bound method in discrete optimization problems", *Autom. 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] Ye. V. Alekseyeva. *Construction of mathematical models of integer linear programming. Examples and problems*, NGU, Novosibirsk, 2012 (in Russian), 132 p.

[5] 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.

[6] J. Kallrath, *Algebraic Modeling Systems: Modeling and Solving Real World Optimization Problems*, Applied Optimization, vol. **104**, Springer Science & Business Media, 2012, 254 p.

[7] T. Koch, T. Ralphs, Y. Shinano. Could we use a million cores to solve an integer program? *Mathematical Methods of Operations Research*, **76**:1 (2012), pp. 67–93.

Sample citation of this publication:

Sergey Smirnov, Vladimir 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”, *Program systems: theory and applications*, 2016, **7**:1(28), pp. 29–46. (*In Russian*).

URL: http://psta.psiras.ru/read/psta2016_1_29-46.pdf