УДК 519.658

О РАСПАРАЛЛЕЛИВАНИИ ЛОКАЛЬНОГО ЭЛИМИНАЦИОННОГО АЛГОРИТМА

© Д. В. Лемтюжникова, О. А. Щербина

Таврический национальный университет им. В.И. Вернадского
факультет математики и информатики
пр-т Вернадского, 4, г. Симферополь, 95007, Украина
e-mail: *darabbt@gmail.com, oshcherbina@gmail.com*

Abstract. We introduce strategies for parallelizing a sequential local elimination algorithm for sparse discrete optimization problems. The local elimination procedure exploits the structure of the underlying problem graph using extended elimination tree. We propose to use hybrid Master-Worker scheme where worker processors (GPUs) solve concurrently subproblems corresponding to super-nodes of extended elimination tree that are generated by a single master process (CPU). Subproblem generation is embedded into an local elimination algorithm and takes previous subproblem solutions into account. The current state of parallel architectures and parallelization technics is discussed.

Введение

Задачи дискретной оптимизации (ДО) с блочной структурой возникают естественным образом во многих приложениях. При этом, задачи ДО являются NP-трудными, поэтому при больших размерах их решение затруднительно в связи с перебором экспоненциального числа вариантов.

Перспективными методами структурной декомпозиции, использующими разреженность матрицы ограничений задач ДО, представляются локальные элиминационные алгоритмы [5], включающие локальные алгоритмы декомпозиции [1], [2, 4, 3], алгоритмы несериального динамического программирования (НСДП) [8, 17, 25], алгоритмы сегментной элиминации [10]. Высокая вычислительная сложность локального алгоритма при больших сепараторах между блоками [5] может быть снижена с помощью параллельных вычислений.

1. Параллельные алгоритмы для задач дискретной оптимизации

Для параллельной реализации алгоритмов ДО могут использоваться системы с распределенной памятью: их параллельное программное обеспечение использует декомпозицию решаемой задачи на подзадачи и назначение их процессорам, а также быструю коммуникационную сеть, обеспечивающую синхронный обмен данными между процессорами. Программирование такой системы может осуществляться с помощью стандартного языка высокого уровня типа C++ или Fortran с явной передачей сообщений. Также может быть использована параллельная система ПО для

решения задач смешанного целочисленного линейного программирования (ЦЛП) — SYMPHONY [24], подобная COIN/BCP [23]. COIN/BCP и SYMPHONY объединены в новом решателе ALPS [22].

Ранее при разработке параллельных алгоритмов использовались обычно суперкомпьютеры и кластеры рабочих станций.

Остановимся на новых возможностях применения современных параллельных вычислительных архитектур для ускорения работы алгоритмов ДО, таких как графические процессоры (GPU) и GRID.

В последнее десятилетие наблюдается повышенный интерес к GRID-вычислениям [11]. GRID-вычисления используют совокупность слабосвязанных разнородных вычислительных ресурсов как единый вычислительный ресурс, которым может пользоваться большое число пользователей.

В течение последних нескольких лет все более популярным в научном сообществе становится использование графических процессоров GPU[1]. Графические процессоры GPU, обычно предназначенные для компьютерной графики, могут использоваться для выполнения расчётов в приложениях, которые обычно проводятся центральным процессором CPU [20]. Среди программных продуктов, которые позволяют программировать для GPU независимо от платформы, можно упомянуть следующие: CUDA[2], DirectCompute[3] от Microsoft, OpenGL Shading Language (GLSL)[4].

Khronos Group разработала специализированный язык OpenCL[5], который является основой для написания программ, выполняемых на разных платформах, состоящих из CPU и GPU. OpenCL является открытым стандартом, который может быть использован для программирования CPU, GPU и других устройств различных производителей, в то время как CUDA специализирована для NVIDIA GPU. OpenCL обеспечивает переносимость различных аппаратных GPU, ОС, программного обеспечения, а также поддерживается многоядерными процессорами. Поэтому представляется перспективным использование именно OpenCL для реализации алгоритмов структурной декомпозиции разреженных задач ДО.

Существенное ускорение параллельных вычислений можно получить с помощью программируемых графических процессоров (GPU), которые могут обрабатывать

[1]GPU — Graphics Processing Unit.

[2]CUDA = Compute Unified Device Architecture.

[3]DirectCompute специализирован исключительно под Microsoft Windows.

[4]GLSL не перспективен с научной точки зрения.

[5]OpenCL (Open Computing Language) — открытый язык вычислений, см. OpenCL: открытый стандарт для параллельного программирования гетерогенных систем. http://www.khronos.org/opencl.

тысячи потоков данных. Среди публикаций, посвященных параллельным алгоритмам ДО на базе GPU, отметим следующие работы. Обзор [21], посвященный исследованиям параллельных алгоритмов муравьиных колоний (АМК)[6] содержит анализ 106 научных публикаций. Можно отметить, что для параллельной реализации АМК часто используется парадигма «Мастер-рабочий»[7] в основном из-за концептуальной простоты и легкости в применении. В [15, 16] рассмотрены особенности параллельной реализации алгоритмов локального поиска и эволюционных алгоритмов на базе GPU. Boyer и соавторы [9] предложили параллельную реализацию метода динамического программирования для задачи о ранце с помощью CUDA для системы из нескольких GPU. Fujimoto и Tsutsui [12] разработали параллельный решатель задачи о коммивояжере для GPU. Gulati и Khatri [13] внедрили новый подход упорядочивания переменных в процедуре решения задачи выполнимости на GPU. Beckers и соавторы [7] предлагают параллельный решатель задачи выполнимости SAT, реализованный на языке OpenCL на GPU.

В [14] описывается распараллеливание метода ветвей и границ (ВГ). Последовательный алгоритм ВГ заключается в неявном переборе подмножеств допустимых решений, с помощью дерева поиска ВГ, вершины которого являются множествами решений рассматриваемой задачи. Размер дерева, то есть количество создаваемых вершин, непосредственно связан с методом, используемым для его построения. Для алгоритма ВГ известны два вида стратегий распараллеливания:

1. Вершинные стратегии, цель которых — ускорить вычисления на уровне вершины дерева поиска, например, параллельное вычисление нижней или верхней границы, параллельная оценка вершин-потомков и т.д.
2. Древовидные стратегии, цель которых — параллельное построение и исследование дерева поиска ветвей и границ.

При распараллеливании алгоритма ВГ можно отметить следующие сложности: структура дерева поиска и граф зависимостей между задачами заранее неизвестны, задачи для процессоров создаются динамически в процессе работы алгоритма. Для того, чтобы избежать дополнительных временных затрат, необходимо принимать во внимание такие алгоритмические аспекты, как балансировка нагрузки и передача сообщений между процессорами.

Хотелось бы отметить работы [6] и [18], [19], посвященные вопросам разработки параллельных версий алгоритмов структурной декомпозиции для дискретных задач.

[6]АМК является метаэвристикой для решения задач оптимизации.

[7]Master-Worker.

Использование ЛЭА при решении задач ДО с большими перемычками (сепараторами) затруднительно в связи с большим объёмом вычислений. Поэтому разработка параллельной версии локального блочно-элиминационного алгоритма для задач ДО представляет значительный интерес.

2. Блочно-элиминационный локальный алгоритм

2.1. **Формулировка локального блочно-элиминационного алгоритма.** Рассмотрим задачу ЦЛП с бинарными переменными:

$$CX = \sum_{j=1}^{n} c_j x_j \to \max \tag{1}$$

при ограничениях

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = 1,\ 2,\ \ldots,\ m, \tag{2}$$

$$x_j = 0,\ 1, \quad j = 1,\ 2,\ \ldots,\ n. \tag{3}$$

Определение 1. Множество переменных, соседних с переменной $x \in X$ в графе $G = (X, E)$, обозначается $Nb_G(x)$ (или $Nb(x)$) и называется *окрестностью* переменной x. *Замкнутой окрестностью* переменной $x \in X$ в графе $G = (X, E)$ называется множество: $Nb[x] = Nb(x) \cup \{x\}$.

Определение 2. *Монотонной* окрестностью вершины x_i называется множество вершин, соседних с x_i, с индексами, большими, чем i (согласно упорядочению α):

$$\overline{Nb}^{\alpha}_{G}(x_i) = \{x_j \in Nb_G(x_i) | j > i\}.$$

Процедура элиминации может быть применена для элиминации не только отдельных переменных, но и множеств переменных в виде «блочной элиминации переменных» [2, 8], которая позволяет элиминировать несколько переменных «блоком». При построении блочных переменных целесообразно использовать понятие *«супер-вершины»*, объединяющей в себе так называемые «неразличимые вершины»[8]. Если задача ДО разбита на блоки, соответствующие подмножествам переменных (называемых супер-переменными), то полученная блочная структура может быть описана с помощью структурного конденсированного графа (фактор-графа), супер-вершины которого соответствуют подмножествам переменных (или блокам) исходного графа и супер-ребра соответствуют соседним блокам. Рассмотрим процедуру локальной

[8]В графе G две смежные вершины u и v называются *неразличимыми*, если $Nb[u] = Nb[v]$.

блочной элиминации [8], в которой элиминация блока (т.е. подмножества переменных) может рассматриваться как элиминация супер-переменной. Рассмотрим упорядоченное разбиение $\mathbf{X}$ множества X переменных на блоки:

$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_p\},\ p \le n,$$

где $\mathbf{x}_l = X_{K_l}$ (K_l — множество индексов, соответствующих $\mathbf{x}_l$, $l = 1, \cdots, p$). Для этого упорядоченного разбиения $\mathbf{X}$, задача ДО P (1.1)–(1.3) может быть решена с помощью ЛЭА, используя *фактор-граф взаимосвязей* $\mathbf{G}$.

При этом $\alpha : \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_p\}$ является порядком элиминации супер-вершин (или блоков) и представляет собой аналог порядка элиминации переменных.

Определение 3. Фактор-граф $\mathbf{G} = (\mathbf{X}, \mathbf{E})$ — это структурный граф супер-задачи ДО, получаемый с помощью представления множества «сжатых» вершин в виде одной супер-вершины и соединения каждой супер-вершины со всеми супер-вершинами, вершины которых были смежны некоторым из сжатых вершин, составляющих рассматриваемую супер-вершину.

Определение 4. Множество супер-переменных, соседних с супер-переменной $\mathbf{x} \in \mathbf{X}$ в фактор-графе $\mathbf{G} = (\mathbf{X}, \mathbf{E})$, обозначается $Nb_{\mathbf{G}}(\mathbf{x})$ (или $Nb(\mathbf{x})$) и называется *окрестностью* супер-переменной $\mathbf{x}$. *Замкнутой окрестностью* супер-переменной $\mathbf{x} \in \mathbf{X}$ в фактор-графе $\mathbf{G} = (\mathbf{X}, \mathbf{E})$ называется множество: $Nb[\mathbf{x}] = Nb(\mathbf{x}) \cup \{\mathbf{x}\}$.

Перейдем к формулировке блочного ЛЭА, состоящего из двух частей.

А. Прямая часть

Рассмотрим вначале блок $\mathbf{x}_1$. Тогда

$$\max_X \{C_N \cdot X_N \mid A_{iS_j} X_{S_j} \le b_i, i \in M, x_j = \{0,1\}, j \in N\} =$$
$$= \max_{X_{K_2}, \cdots, X_{K_p}} \{C_{N-K_1} \cdot X_{N-K_1} + h_1(Nb(X_{K_1})) \mid A_{iS_j} X_{S_j} \le b_i,$$
$$i \in M - U_1, x_j = \{0,1\}, j \in N - K_1\}$$

где $U_1 = \{i : S_i \cap K_1 \ne \varnothing\}$ и

$$h_1(Nb(X_{K_1})) = \max_{X_{K_2}, \cdots, X_{K_p}} \{C_{K_1} \cdot X_{K_1} \mid A_{iS_j} X_{S_j} \le b_i,$$
$$i \in U_1, x_j = \{0,1\}, j \in Nb[x_1]\}$$

Первый шаг локальной блочной элиминационной процедуры состоит в решении, используя полный перебор значений X_{K_1}, следующей задачи оптимизации

$$h_1(Nb(X_{K_1})) = \max_{X_{K_2}, \cdots, X_{K_p}} \{C_{K_1} \cdot X_{K_1} \mid A_{iS_j} X_{S_j} \le b_i,$$
$$i \in U_1, x_j = \{0,1\}, j \in Nb[x_1]\}$$

и запоминании оптимальных локальных решений X_{K_1} в виде функций окрестностей X_{K_1}, т.е. $X^*_{K_1}Nb(X_{K_1})$. Максимизация $f(X)$ по всем возможным присвоениям $Nb(X_{K_1})$ называется *элиминацией блока* (или элиминацией супер-переменной) X_{K_1}. После элиминации необходимо решить задачу оптимизации:

$$\max_{X-X_{K_1}} \{C_{N-K_1} \cdot X_{N-K_1} + h_1(Nb(X_{K_1})) \mid A_{iS_j}X_{S_j} \le b_i,$$
$$i \in M - U_1, x_j = \{0,1\}, j \in N - K_1\}$$

Заметим, что эта задача имеет тот же вид, что и исходная задача, а табличная функция $h_1(Nb(X_{K_1}))$ может быть рассмотрена как новый компонент измененной целевой функции. Следовательно, та же процедура может быть использована для поочередной элиминации блоков – супер-переменных $\mathbf{x}_2 = X_{K_2}, \cdots, \mathbf{x}_p = X_{K_p}$. На каждом шаге j запоминаются новый компонент $h_{\mathbf{x}_j}$ и оптимальные локальные решения $X^*_{K_j}$ в виде функций от $Nb(X_{K_j} \mid X_{K_1}, \cdots, X_{K_{j-1}})$ т.е. от множества переменных, взаимосвязанных по крайней мере с одной переменной из X_{K_j} в текущей задаче, полученной из исходной задачи с помощью элиминации $X_{K_1}, \cdots, X_{K_{j-1}}$. Так как множество $Nb(X_{K_p} \mid X_{K_1}, \cdots, X_{K_{p-1}})$ пустое, элиминация X_{K_p} позволит получить оптимальное значение целевой функции $f(X)$.

В. Обратная часть

Эта часть процедуры состоит в последовательном выборе $X^*_{K_p}, X^*_{K_{p-1}}, \cdots, X^*_{K_1}$ — оптимальных локальных решений из сохраненных в прямой части таблиц $X^*_{K_1}(Nb(X_{K_1}))$, $X^*_{K_2}(Nb(X_{K_2} \mid X_{K_1}))$, $\cdots$, $X^*_{K_p}(Nb(X_{K_p} \mid X_{K_1}, \cdots, X_{K_{p-1}}))$.

Графовой интерпретацией блочного ЛЭА может служить *обобщенная элиминационная игра*, входом которой является фактор-граф взаимосвязей $\mathbf{G}$ и обобщенный порядок элиминации (упорядоченное разбиение) $\alpha : \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_p\}$ супер-вершин графа $\mathbf{G}$. На каждом шаге $\nu (1 \le \nu \le p)$ обобщенной элиминационной игры окрестность $Nb(\mathbf{x}_\nu)$ супер-вершины $\mathbf{x}_\nu$ преобразуется в клику, а $\mathbf{x}_\nu$ удаляется из графа $\mathbf{G}$ вместе со смежными ребрами.

Определение 5. Граф $\mathbf{G}^+_{\mathbf{X}} = (\mathbf{X}, \mathbf{E}^+)$, образованный путем добавления в $\mathbf{G}$ всех ребер, добавленных алгоритмом, называется *обобщённым пополненным графом.*

Понятие монотонной окрестности, введенное выше, может быть обобщено на случай супер-переменной в фактор-графе.

Определение 6. *Монотонной* окрестностью супер-вершины $\mathbf{x}_i$ в фактор-графе $\mathbf{G} = (\mathbf{X}, \mathbf{E})$ называется множество супер-вершин, соседних с $\mathbf{x}_i$ в фактор-графе $\mathbf{G}$, с индексами, большими, чем i (согласно упорядочению $\alpha : \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_p\}$):

$$\overline{Nb}^{\alpha}_{\mathbf{G}}(\mathbf{x}_i) = \{\mathbf{x}_j \in Nb_{\mathbf{G}}(\mathbf{x}_i) | j > i\}.$$

Бесконтурный орграф вычислительной процедуры локального блочного элиминационного алгоритма содержит вершины, соответствующие вычислению функций $h_{\mathbf{x}_j}(Nb_{G_{\mathbf{X}}^{(j-1)}}(\mathbf{x}_j))$ и является *обобщенным элиминационным деревом.*

Определение 7. Обобщённым элиминационным деревом (ОЭД) графа $\mathbf{G}$ для упорядоченного разбиения множества вершин на подмножества (супер-вершины) называется ориентированное дерево $\vec{\mathbf{T}}_{\mathbf{X}}$, имеющее то же множество супер-вершин $\mathbf{X}$, что и фактор-граф $\mathbf{G}$, а множество дуг ОЭД определяется с помощью отношения «отец–сын» следующим образом: отцом супер-вершины $\mathbf{x}$ является первая супер-вершина из монотонной окрестности $\overline{Nb}_{\mathbf{G}_{\mathbf{X}}^{+}}(\mathbf{x})$ супер-вершины в пополненном графе $\mathbf{G}_{\mathbf{X}}^{+}$.

3. Источники распараллеливания

Для ЛЭА структура элиминационного дерева известна заранее и граф зависимостей между задачами предсказуем, что позволяет избежать ряда описанных в разделе 1 проблем, свойственных параллельным методам ВГ.

Для ЛЭА при решении задачи (1)–(3) существуют следующие источники распараллеливания:

1) *Параллельная обработка независимых ветвей обобщенного элиминационного дерева.*

Используемый граф потоков данных для блочного ЛЭА задачи является обобщенным элиминационным деревом, которое должно обрабатываться от листьев к корню. При параллельной реализации блочного ЛЭА супер-вершины каждого слоя ОЭД могут быть рассмотрены и обработаны параллельно. При расчетах множество промежуточных данных передается в родительскую вершину из вершин-сыновей. Как только промежуточная информация от всех вершин-сыновей собрана в родительской вершине, она используется при решении соответствующей родительской вершине задачи ДО. Таким образом, независимые ветви дерева могут обрабатываться параллельно. Распараллеливание ЛЭА более эффективно в нижней части дерева, нежели вблизи ее корневой вершины.

2) *Параллельное решение задач в блоках.*

Для каждого блока (супер-вершины) можно рассмотреть множество независимых задач[9], которые порождаются в прямой части ЛЭА. Там для каждого значения сепаратора $\mathbf{X}_{Nb_{G_{\mathbf{X}}^{(j-1)}}(\mathbf{x}_j)}$ нужно решить параллельно задачи $h_{\mathbf{x}_j}(Nb_{G_{\mathbf{X}}^{(j-1)}}(\mathbf{x}_j))$.

[9]Независимость задач состоит в том, что информация, полученная в результате решения одной задачи, не используется при решении другой.

Так как структура графа потоков данных известна (ОЭД)[10], такие задачи могут быть решены при помощи парадигмы «Master-Worker», в которой имеются два различных типа процессоров: мастер-процессор и рабочие процессоры. При этом управление алгоритмом осуществляется мастер-процессором, который разбивает задачу на отдельные подзадачи и назначает их для решения рабочим процессорам, учитывая взаимосвязи между ними. Мастер-процессор выбирается в начале решения пакета задач, соответствующих блоку ОЭД. Далее, рабочие процессоры выбираются мастером динамически из списка процессоров-кандидатов, используя критерий хорошей балансировки нагрузки.

Для параллельной реализации ЛЭА мы предлагаем использовать гибридную схему «мастер-рабочий», которая допускает одновременное использование CPU и GPU. GPU, множество основных параллельных машин с общей памятью, выступают в качестве рабочих процессоров и выполняют решение подзадач ДО для блоков. Синтез решений выполняется процессором CPU, который выступает в роли мастера.

Заключение

В работе выделены стратегии распараллеливания локального элиминационного алгоритма для разреженных задач дискретной оптимизации на основе использования современных вычислительных архитектур. Независимые подзадачи, соответствующие разным блокам, и подзадачи, соответствующие независимым ветвям обобщенного элиминационного дерева, могут решаться параллельно при помощи таких вычислительных архитектур, как многоядерные процессоры, графические процессоры (GPU) и GRID. Для параллельной реализации ЛЭА предлагается использовать гибридную схему «мастер-рабочий», которая допускает одновременное использование CPU и GPU, причем GPU, множество основных параллельных машин с общей памятью, выступают в качестве рабочих процессоров и выполняют решение подзадач ДО для блоков, а CPU используется в качестве мастера. В дальнейшем планируется реализовать эту схему при помощи виртуального программирования, что позволит оценить эффективность распараллеливания локального элиминационного алгоритма и определить дальнейший ход исследования.

Список літератури

1. Журавлев Ю. И. Избранные научные труды / Ю. И. Журавлев. — М.: Магистр, 1998. — 420 с.
2. Щербина О. А. О локальных алгоритмах решения квазиблочных задач дискретного программирования / О. А. Щербина // Проблемы кибернетики. — М.: Наука, 1983. — Вып. 40. — С. 171–200.

[10]В качестве графа потоков данных может использоваться и древовидная декомпозиция.

3. Щербина О. А. О несериальной модификации локального алгоритма декомпозиции задач дискретной оптимизации / О. А. Щербина // Динамические системы. — 2005. — Вып. 19. — С. 179–190.
4. Щербина О. А. О решении квазиблочных задач целочисленного линейного программирования с помощью локального алгоритма / О. А. Щербина // Кибернетика. — 1984. — № 2. — С. 118–121.
5. Щербина О. А. Локальные элиминационные алгоритмы решения разреженных дискретных задач / О. А. Щербина // Журнал вычислительной математики и математической физики. — 2008. — Т. 48, № 1. — С.161–177.
6. Allouche D. Towards parallel non serial dynamic programming for solving hard weighted CSP / David Allouche, Simon De Givry, and Thomas Schiex // Proceedings of the 16th international conference on Principles and practice of constraint programming (CP'10), David Cohen (Ed.). — Berlin, Heidelberg: Springer-Verlag, 2010. — P. 53-60.
7. Parallel SAT-solving with OpenCL. [Электронный ресурс] / [Beckers S., De Samblanx G., De Smedt F. et al.]. — 2011. Режим доступа: http://hgpu.org/?p=5769.
8. Bertele U. Nonserial Dynamic Programming / U. Bertele, F. Brioschi. — New York: Academic Press, 1972. — 235 p.
9. Boyer V. Solving knapsack problems on GPU / V. Boyer, D. Baz El, M. Elkihel // Comput. Oper. Res. — 2012. — 39 (1). — P. 42–47.
10. Dechter R. Bucket elimination: A unifying framework for reasoning / R. Dechter // Artificial Intelligence. — 1999. — V. 113. — P. 41–85.
11. Foster I. The Grid: Blueprint for a New Computing Infrastructure / I. Foster, C. Kesselman. — San Francisco: Morgan Kaufmann Publishers Inc, 2002. — 677 p.
12. Fujimoto N. A highly-parallel TSP solver for a GPU computing platform / N. Fujimoto, S. Tsutsui // Proc. of the 7th int. conf. on Numerical methods and applications (NMA'10) // I. Dimov, et al. (Eds.). — Springer-Verlag, Berlin, Heidelberg, 2010. — P. 264–271.
13. Gulati K. Boolean Satisfiability on a Graphics Processor / K. Gulati, S. P. Khatri // ACM Great Lakes Symposium on VLSI 2010. — P. 123–126.
14. Le Cun B. Parallel branch and bound algorithms / B. Le Cun, C. Roucairol, T.G. Crainic // Parallel Combinatorial Optimization, chapter 1. — USA: John Wiley and Sons, 2006. — P. 1–28.
15. Luong T-V. Parallel Hybrid Evolutionary Algorithms on GPU / T-V. Luong, N. Melab, E-G. Talbi // IEEE Congress on Evolutionary Computation (CEC). — Barcelone. — 2010.
16. Luong T-V. GPU-based Multi-start Local Search Algorithms / T-V. Luong, N. Melab, E-G. Talbi // Proceedings of Learning and Intelligent Optimization (LION'2011). — Rome, Italy, 2011. — P. 321–335.
17. Neumaier A. Nonserial dynamic programming and local decomposition algorithms in discrete programming [Электронный ресурс] / A. Neumaier, O. Shcherbina. Режим доступа: http://www.optimization-online.org/DB_HTML/2006/03/1351.html.
18. Otten L. Towards parallel search for optimization in graphical models / L. Otten, R. Dechter // Proc. of ISAIM 2010. — Fort Lauderdale, USA, 2010.
19. Otten L. Advances in distributed branch and bound / L. Otten and R. Dechter // Proceedings of ECAI'12, Montpellier, France, August 2012.

20. A survey of general-purpose computation on graphics hardware / [Owens J.D., Luebke D., Govindaraju N. et al.] // State of the Art Reports, August 2005. — Eurographics, 2005. — P. 21–51.
21. Pedemonte M. A survey on parallel ant colony optimization / M. Pedemonte, S. Nesmachnow, H. Cancela. // Applied Soft Computing. — 2011. — V. 11. — P. 5181–5197.
22. Ralphs T. K. A library hierarchy for implementing scalable parallel search algorithms / T. K. Ralphs, L. Ladanyi, M. J. Salzman // Journal of Supercomputing. — 2004. — V. 28(2). — P. 215–234.
23. Ralphs T. K. COIN/BCP User's Guide [Электронный ресурс] / T. K. Ralphs, L. Ladanyi. — 2001. Режим доступа: http://www.coin-or.org.
24. Ralphs T. K. Parallel branch, cut and price for large-scale discrete optimization / T. K. Ralphs, L. Ladanyi, M. J. Salzman // Mathematical Programming. —2003. — B. 98. — P. 253–280.
25. Shcherbina O. Nonserial dynamic programming and tree decomposition in discrete optimization / O. Shcherbina // Proceedings of Int. Conference on Operations Research "Operations Research 2006" (Karlsruhe, 6-8 September, 2006). — Berlin: Springer Verlag, 2007. — P. 155–160.

Статья поступила в редакцию 02.06.2012